MASTER'S THESIS

Convolutional Neural Networks on the Edge

Stremerch, K.J.P.

Award date:
2021

CONVOLUTIONAL NEURAL NETWORKS ON THE EDGE

Kurt Stremerch

Student:	Kurt Stremerch
Student number:	
Date of presentation:	July 9th, 2021

Open Universiteit
de beste! www.ou.nl

CONVOLUTIONAL NEURAL NETWORKS ON THE EDGE

A master thesis submitted by

Kurt Stremerch

Open University of the Netherlands, Faculty of Science
Master's Programme in Software Engineering

Student:	Kurt Stremerch	
Student number:		
Course:	IM9906	
Date of presentation:	July 9th, 2021	
Thesis committee:	dr. Arjen Hommersom (chairman),	Open University
	dr. Martijn van Otterlo (supervisor),	Open University
	dr. Joeri Verbiest (supervisor),	Karel De Grote Hogeschool

CONTENTS

Convolutional Neural Networks

On the edge

Kurt Stremerch

Abstract

The use of machine learning on IoT data has opened up lots of opportunities. Neural networks are used to analyze the data and make sense of it by converting data into useful information in real-world applications such as speech recognition or image classification. Today, optimized neural networks found their way from the cloud to IoT devices. These high-end embedded devices are much more powerful than tiny embedded devices in wearables or implanted medical devices. This research aims to investigate to which extent convolutional neural networks can be used on tiny embedded systems in the context of audio classification. Three challenges regarding a cochlear implant application have been considered; hardware resource limitations, the model type versus nature of sounds to classify, and the impact of subcutaneous MEMS microphone. From a wide range of experiments, we have learned that post quantization and quantization aware training models can score equally well on the UrbanSound8k dataset compared to floating point models. Acoustic event detection models can characterize an acoustic environment where the scene classification score can be improved by transferring knowledge from an event classification task. The simulated subcutaneous recordings performed poor on all features still the Mel feature achieved the highest classification score. This research shows that convolutional neural networks for audio classification can be effectively reduced in size to make it suitable for tiny embedded devices, however, the edge hardware specifications must be taken into account. The frequency-specific output of an embedded microphone can lead to significant accuracy loss during deployment.

1. INTRODUCTION

Over the past years, millions of small embedded devices have been deployed to build the Internet of Things. The Internet of Things is simply an extension of the current internet, further into our physical world, into things. These devices typically have a sensor function and send the acquired data to the cloud for further processing. The use of machine learning on the collected data has opened up lots of opportunities. Neural networks are used to analyze the data and make sense of it by converting the data into useful information. Deep learning and neural networks represent powerful techniques to support applications in speech recognition, image classification or other real-world problems (Wang et al. [2020], Alzubaidi et al. [2020], Saleem and Khattak [2019]). In the past years, open-source frameworks created an ecosystem for researchers that generalize the coding of neural networks. Using graphical processor units (GPU) to accelerate the deep learning frameworks, researchers and data scientists can significantly speed up deep learning training.

An example of a real-world problem can be found with people with profound hearing loss. These people can not hear anymore with regular hearing aids and require a cochlear implant. A cochlear implant is a surgically implanted device that bypasses normal acoustic hearing. It stimulates the auditory nerves electrically based on the audio picked up by a microphone. A person could be listening to music, having a conversation at home, or walking through a crowded street. Depending on the acoustic scene, the hearing device optimizes its acoustic settings to interact better with the environment. It reduced the listening effort and fatigue associated with hearing loss. Another problem can arise with sounds that are new to the brain. When the implant is activated after surgery, some sounds are heard for the first time. During the therapy, the brain learns to adapt to the new stimulations. More specific sounds, like running water over skin, can be new and uncomfortable for some people. The detection of particular sound events can help to increase the comfort of people with a cochlear implant.

The analysis of acoustic sounds is called audio classification. The audio classification algorithms are usually hand-engineered algorithms that require a high level of audio domain knowledge. The computation is mostly handled by a dedicated digital signal processor (DSP). The program code is small, highly optimized, and parallelized. Adding a new audio scene to the classification could require a redesign of the algorithm and program code.

Machine learning and neural networks have also proven to handle audio classification very well. Instead of coding the algorithm, the neural network learns the algorithm by providing the audio data and the audio scene label information during the training process. Adding a new audio scene would require retraining the network, including the extra data. Neural networks are more flexible to requirement changes than hand-engineered algorithms and could provide a more generic solution for classification problems.

In the past years, a new trend in machine learning research has focused on neural network efficiency. Today, optimized neural networks found their way from the cloud to IoT devices. These high-end embedded devices are much more powerful than tiny embedded devices in wearables or implanted medical devices, for example, cochlear implants (CI).

Before neural network models can be deployed on ultra low power devices, like CIs, they need to be optimized even more to fit the reduced memory size and match the computational power of the resource-scarce devices. Always-on devices are active for a short time to execute their task and hibernate back to an ultra low power state. During the active time, the device executes parts of the application code in volatile memory to prevent executing code from slower and more power-consuming non-volatile memories. The volatile memory is more limited than non-volatile memory on microcontrollers. This research investigates the impact on accuracy using quantization optimization techniques on already highly optimized small neural networks in the context of audio classification. Quantization reduces the arithmetic precision of neural networks to reduce the memory size and computational processing time.

The document is organized as followed: Sections 2, 3 and 4 provide background information. The research questions and research method are formulated in Section 5. Section 6 reports the results and discussion. Section 7 concludes the thesis.

2. Background

This section provides background information on embedded devices and audio classification. Section 2.1 introduces embedded devices and divides them into high-, low- and tiny end categories based on the resource constraints. Section 2.3 introduces acoustic classification tasks together with the publicly available acoustic datasets and three data features that can be extracted.

2.1. Embedded devices

Embedded devices are small computer systems that are designed to perform a dedicated function. The hardware is made to measure depending on the application. Embedded devices are everywhere in our daily life, from game consoles to microwaves or from traffic cameras to wearables.

The core component of an embedded system is a microcontroller unit (MCU) which consists of a processing unit, memory, and peripheral blocks. The difference with computer processors is that the computational power is an order of magnitude six lower. Microcontrollers also have onboard hardware interfaces that allow them to interact with other chips like sensors and actuators. The software that runs on the hardware is called firmware. The program code is highly optimized due to memory constraints. The firmware functions can access the hardware or peripherals immediately without help from an operating system. The overhead is highly reduced at the cost of flexibility.

Microcontrollers can be divided into roughly three groups. The high-end microcontrollers (e.g. Cortex-A) have high computational power and available memory. They are used for graphical user interface applications like point of sales, cameras, or car entertainment systems. The low-end microcontrollers (e.g. Cortex M4 or higher) are often found in IoT devices like smart thermostats or home assistants. The memory and computational power are limited which drives the cost down. Both high and low-end devices are always considered on systems where an uninterruptable power source is available. The

third group are the tiny microcontrollers that target battery-powered devices. Even though the resources are even more constrained on these devices, the available resources are disabled as much as possible to preserve battery life. A peripheral is only activated when an action is required and powered off when the device is sleeping until the next action.

One of the drivers for power optimization is the usage of memory. The microcontroller has two types of memory. The flash memory is non-volatile memory where the firmware is programmed. The RAM is volatile memory, where the variables are stored during runtime. The flash memory access latency is slower and more power-consuming than the RAM. To extend the autonomy of tiny battery-powered devices even further, some firmware applications will store a part of the program code in the RAM, which is always powered and active. When the device wakes up, the code execution in RAM memory will be faster which allows the device to go back to sleep faster. There is also no need to power up the flash memory. The firmware code instructions in RAM will decide, based on the sensor data, if code execution in flash is required. Only then the more power-consuming flash memory is activated and accessed. Even though flash instruction cache logic exists to accelerate flash access on low-end devices, executing code from flash still consumes more because the flash memory is powered from a higher voltage than the processor core voltage. The downside is that the RAM size is much smaller than the flash size on most microcontroller parts. Adding external memory to the microcontroller will drive costs and lead to a considerable increase in power consumption (Zhang and Kouzani [2019]). Loading a 32bit value from on-chip SRAM can save 120x the energy required to fetch it from external DRAM according to Han et al. [2016]. Battery-powered implanted medical devices are tiny embedded devices that have considerable space and power constraints. The SRAM memory for the tiny microcontroller devices is limited to 128kB.

One of the most important differences between high-end and lower-end microcontrollers is the presence of a floating-point unit (FPU). This silicon block inside the processor unit is capable of executing floating-point math in hardware. This means that the compiled code of a high-end microcontroller will use hardware instructions to command the FPU to perform floating-point operations. The low-end microcontroller firmware compiler will need a firmware-based library implementation to generate instructions for the processor to calculate the floating-point operation using integer values. This firmware based floating-point calculation comes with a performance penalty for low-end microcontrollers.

2.2. DIGITAL SOUND

Sound is a vibration that travels through a medium as a pressure wave. It can be heard by humans when the pressure changes vibrate the eardrum and stimulate the auditory nerve. The wave can be converted from a mechanical wave into an electrical signal by using a microphone. Next, the continuous analog signal is converted into a set of snapshots or samples. A sample records the strength indication of the electrical signal. The samples are taken at regular time intervals which is called the sample rate. An analog to digital converter converts the sample amplitude information into binary data. Figure 1 illustrates a digital sound wave. The number of wave repetitions per second is the frequency measured in Hertz (Hz). A sound can be a combination of multiple waves with different amplitude and

frequency information over time. The frequency information (Figure 2) can be extracted by computing the Fast Fourier Transform (Good [1997]). It is common to analyze the spectral content over time. The sound is split up into chunks where the frequency spectrum of each chunk is estimated. This time-frequency representation results in a spectrogram as illustrated in Figure 4.

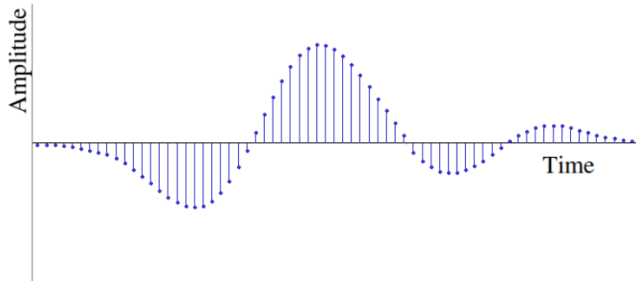

Figure 1: Sampled sound wave

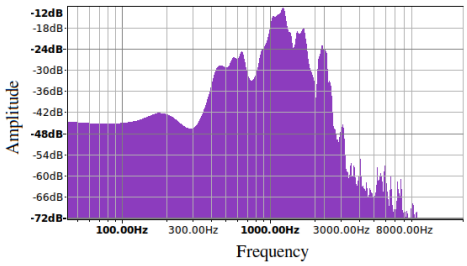

Figure 2: Frequency spectrum

Frequencies that humans can hear is called audio. The hearable audio range of the human ear is 20Hz to 20kHz. The amplitude sound pressure level (SPL) ranges from 20uPa to 20Pa. Therefore the amplitude is usually represented using a logarithmic scale. The loudness perceived by normal human hearing is most sensitive in the 2kHz-5kHz range (Eargle [2002]). The sensitivity of a microphone depends on the type and construction of the transducer element. Nowadays, modern microphones are built on chip level by using microelectromechanical systems (MEMS) technology that mounts pressure-sensitive elements on silicon.

2.3. AUDIO CLASSIFICATION

Audio classification is the process of analyzing sound and deciding to which of a set of classes the sound most probably belongs. The method extracts relevant features from the audio signal and uses these features to characterize the sound. Generally, audio classification tasks can be divided into three subdomains. The first subdomain is the most known type of audio classification called speech recognition. All modern cars have voice-activated functions. The driver dictates the destination address to the car GPS system. The system understands the language and is able to differentiate dialects and intonations. A second subdomain is music classification. The sounds are generated by instruments and have a distinct tempo, rhythm, and note patterns. The intention is to classify the music according to the genre or instruments played. The third subdomain addresses the acoustic scene and event classification. The latter is the group of interest for this thesis.

Imoto [2018] has reviewed the basics in acoustic event and scene analysis. The acoustic scene classification (ASC) and acoustic event classification (AEC) are two tasks within the audio classification. The goal of ASC is to recognize the audio environment based on the physical or social context. Indoor and outdoor sounds are recognized and consolidated into a scene like an office room or public place. The goal of AEC is the recognition of individual sound events. These specific sounds could be running water, scratching hair, or a siren. For both classification tasks, the sound input is often marked as a clip. In ASC networks, the sound clip is relatively long, ranging from seconds to tens of seconds. The clip contains multiple sound events that contribute to the scene characteristic. For AEC, the in-

put audio clip is relatively short, ranging from tens of milliseconds to several seconds. The clip contains only one event that is classified with a single label output.

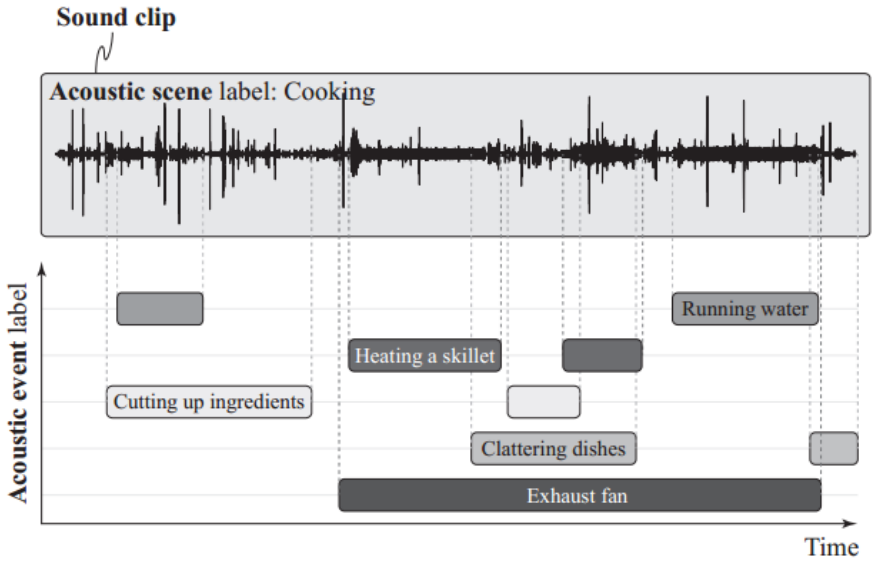

Figure 3: Terms used in event and scene anaylsis (Imoto [2018])

Abeßer [2020] has reviewed deep learning-based methods for ASC. A more specific classification group within ASC is the detection of temporarily found events within the acoustic scenes. For example, urban sounds can contain car horns, sirens, pedestrians walking, etc. ASC is an active research domain stimulated by the Detection and Classification of Acoustic Scenes and Events (CASE) competition. This annual competition shows the current state-of-the-art for ASC. Mesaros et al. [2017b] has shown that humans perform noticeably worse than the deep learning solutions published within the DCASE competition. The top 10 state-of-the-art publications are convolutional neural network (CNN) based architectures. The convolutional aspect of CNNs is explained in section 3. CNN models use convolution and pooling layers for feature extraction which are typically input into dense (fully-connected) layers for classification. Sigtia et al. [2016] has demonstrated that fully-connected layers require the least number of operations and achieve the best performance when compared to other classification types like vector machines and Gaussian mixture models.

Although CNN is typically used for image recognition and classification, it can be used for audio classification by transforming the audio information into an image. This requires a preprocessing step where the audio pressure wave signal is converted into a different data format before it is input to the convolutional layer. The review highlights two types of signal transformations relevant for CNN ASC; learnable signal transformation and fixed signal transformation. The learnable signal transformation models process the raw data directly. The audio signal is chopped into a time frame and fed to the neural network. The frequency information is extracted using learnable filters during the training process. Lee et al. [2017] presented a raw waveform-based audio classification using sample-level CNN architectures. Their ReSe2Multi model only performed 2% less compared to the state-of-

the-art in the DCASE 2017 challenge. The fixed signal transformation uses a mathematical transformation independent of the input data. The transformation is first applied to the audio data before it is presented to the input of the CNN. Hussain et al. [2017] reviewed the DCASE 2017 submissions and concluded that the research for spectrogram, MFC and CQT features (section 2.3.2) is the way forward to enhance the accuracy of the DCASE challenges. The most popular transformations in the DCASE 2019 are the short time Fourier transform (STFT), the Mel spectrogram and the wavelet spectrogram.

Rank	Code	Accuracy	Model size
1	Zhang IOA task1a 3	85.2	192MB
18	Zeinali BUT task1a 3	79.1	100MB
26	LamPham HCMGroup task1a 1	76.8	48MB
31	McDonnell USA task1a 3	80.4	18MB
40	McDonnell USA task1a 2	80.5	12MB
41	Wilkinghoff FKIE task1a 1	74.6	4MB
48	Liang HUST task1a 2	66.4	1.2MB
70	DCASE2019 baseline	62.5	464kB

Table 1: Selection of DCASE 2019 submissions (appr. 50% model complexity decrease)

Although the state of the art ASC models are mature, Abeßer [2020] discusses two novel challenges that arise during real-world deployment of audio classification models. The first challenge refers to audio recording quality. Mobile edge devices have space constraints and use micro-electro-mechanical devices to capture the audio. The recording conditions are different from the high-quality recorded scientific datasets. The urban acoustic dataset of Mesaros et al. [2018], used for the DCASE challenge, was recorded using high-quality microphones. Since the 2019 DCASE challenge (Mesaros et al. [2019b]) there is a new task for ASC with mismatched recording devices. The second challenge mentioned is model complexity. Due to privacy restrictions, audio conversations can not be recorded without consent. The recorded voice data is considered personal data, which is protected under EU law. Instead of sending the voice audio to the cloud, the audio is kept at sensor device level and classified locally. The resulting classification is anonymous and can be used. Local classification requires a model compression where the size of the model is reduced and redundant information in the audio data needs to be identified. Table 1 lists a reduced set of submissions for the DCASE2019 competition based on a 50% model size difference step, starting from the winning submission. The model sizes are too large to deploy on tiny embedded devices (128kB memory limit). Hence the 2020 DCASE challenge Heittola et al. [2020] published a new task in the research world to address this subject in the 'low-complexity acoustic scene classification task'.

2.3.1. Acoustic datasets

Datasets are large collections of data used for machine learning. High quality-labeled training sets are expensive to produce because it very time consuming to manually label large amounts of data. Speech and music sounds are relatively easy to collect as human actions generated them. Collecting environmental sounds is less evident to capture as multiple

sound events overlap during the recording. The sound clips are annotated by labels to indicate which events and scenes are heard. The interpretation of the audio by human listeners can be different for the same sound clip. This has driven the generation of public datasets. When developing new algorithms for sound classification, a public dataset is essential. A common dataset allows researchers to compare methods because the reference is identical. The following items list public datasets used in the context of audio classification:

- Imoto [2018] describes the public datasets available for acoustic event and scene analysis. The RWCP Sound Scene database in Real Acoustic Environments published by Nakamura et al. [2000] contains dry sound of 105 types of acoustic event such as 'spay' or 'clapping', recorded in 0.5 to 2s audio clips. Foster et al. [2015] created the CHiME-Home dataset for the CHiME corpus challenge by Christensen et al. [2010]. The dataset was created to develop speech separation in a domestic environment and contains 6100 audio clips of 4s in nine classes such as child speech, adult male speech, video/tv sounds. Other described datasets like the TUT acoustic scenes and events DCASE dataset and ESC dataset are mentioned below.

- Gemmeke et al. [2017] created the AudioSet which consists of 632 audio event classes based on 2.084.320 sound clips. The goal was to create a general-purpose audio database. The clips are 10 second long sound files originally from YouTube videos. The events are human-labeled ranging from vehicle sounds, natural sounds, music, animal sounds, domestic sounds, etc..

- Dekkers et al. [2017] created the SINS dataset of live recordings in a home environment. The recording was executed using a sensor network spread over different rooms of a vacation home. One person lived in the environment for a week. The dataset contains 9 audio scenes which are categorized in eating, cooking, watching TV, vacuum cleaning, etc..

- Salamon et al. [2014b] has created the Urbansound8k dataset containing 8732 labeled recordings of a maximum 4 seconds long. The dataset covers 10 classes like a siren, children playing, dog bark, etc.. in a street environment. Later, Salamon et al. [2017] has used Scraper, an open-source python library for soundscape and data augmentation to augment the Urbansound8k dataset to 50.000 recordings. The current state of the art accuracy is 79% by Salamon and Bello [2016]. on the Urbansound8k is and .

- The Piczak [2015a] ESC50 dataset contains 50 classes in 2000 labeled recordings and 250.000 unlabeled recordings. The high-level categories are animal sounds, natural soundscapes and water sounds, human (non-speed) sounds, domestic sounds, and urban sounds. Each high-level category contains 10 subclasses. This dataset is also classified by human listeners.

- Mesaros et al. [2017a] has created the DCASE 2017 dataset where all samples were recorded using binaural Soundman OKM II Klassik/studio A3 electret in-ear microphones. The dataset consists of three sub-datasets for scene classification, event classification, and rare sound event detection. The acoustic scene dataset consists of 15 acoustic scene classes like a park, home, or office. Each scene contains 312 recordings of 10 seconds. The acoustic event dataset consists of 6 classes (car, people speaking, people walking, large vehicle, and brakes squeaking) with 659 events found in clips of 3 to 5 minutes. This set contains multiple temporally overlapping events. The DCASE 2019 dataset has 40 hours of recording using 10 acoustic scenes. The submission of Chen et al. [2019] is the current state of the art with 85.2% accuracy on the DCASE 2019 dataset.

- Bach [2020] has created the Hearing Aid Research Data Set for Acoustic Environment Recognition (HEAR-DS). It contains binaurally recorded clips of speech in different acoustic environments. It contains scenes with and without speech like inTraffic, speechInTrafic, wind, speechInWind, etc.. The 14 classes are represented in 10226 audio clips of 10 seconds where the speech is generated according to Chime2018 dataset of Barker et al. [2018]. The HEAR-DS dataset is created for hearing aids to suppress the noisy acoustic environment of the detected scene.

2.3.2. Data feature

A data feature is a measurable property of data. A feature can be extracted directly or can be constructed using a transformation algorithm to represent the original data differently. Depending on the task, domain knowledge is used to select the features during the feature engineering phase. In the context of audio classification, the time-frequency information is extracted from the data. The network model training can focus on the patterns of the transformation, instead of learning how to do the transformation itself. The architecture of the raw audio processing model is more complex and increases the training time. The developers decide which data representation to use based on the network constraints. The selection is important to achieve an acceptable accuracy, but it also drives the required deployment processing power where the product hardware needs to preprocess the data in the same way before it can be input into the model. The feature extraction represents the data in a smaller way without losing essential information. This reduces the required input size of a model which is beneficial for the memory size and computational complexity of the network model. Three popular input features in the context of audio classification are short time Fourier transform, constant Q transformation and Mel transform.

2.3.2.1 Short time Fourier transform

The short time Fourier transformation or STFT (Allen [1977]) converts an audio signal into time-frequency representation $X[k,m]$ according to (1) using frequency k and time/sample index m. Every audio time fragment $x[n]$ is transformed into a set of energy levels for the frequency bands or bins. The bin center frequencies are linearly spaced. The frequency resolution is defined by the sampling frequency of the signal divided by the

number of samples N. Each time fragment is multiplied by a window $w[n]$ to prevent discontinuities at the begin/end of the fragment. A large window represents a good frequency resolution where frequencies close to each other can be detected, but it results in a poor time resolution where the time where the frequencies change is more difficult to determine. A short time window has a good time resolution but results in a poor frequency resolution.

$$X[k,m] = \sum_{n=0}^{N-1} x[n]\, w[n] e^{-j2\pi kn/N} \tag{1}$$

2.3.2.2 Constant Q transform

The constant Q transformation or CQT (Brown [1991]) is a similar transformation compared to STFT but uses logarithmically spaced center frequencies according to (2). The transformation represents a series of filters f_k, where the k-th filter has a spectral width equal to a multiple of the previous filter width. Because of the varying filter width, the window length of each bin is function of the bin number.

$$X[k] = \frac{1}{N[k]} \sum_{n=0}^{N[k]-1} x[n]\, w[k,n] e^{\frac{-j2\pi Qn}{N[k]}} \tag{2}$$

The Q factor refers to the ratio of the center frequency f_k of versus the bin frequency bandwidth δf_k. The Q factor is kept constant hence the filter bandwidth of low frequency bins is smaller compared to the high frequency bins. This results in a higher frequency resolution at low frequencies compared to the STFT transformation. The number of filters n is defined per octave where the center frequency of the first filter is f_{min} according to (4).

$$Q = \frac{f_k}{\delta f_k} \tag{3}$$

$$\delta f_k = 2^{1/n} \cdot \delta f_{k-1} = \left(2^{1/n}\right)^k \cdot \delta f_{\min} \tag{4}$$

2.3.2.3 Mel transform

The linear frequency spacing is not always desired for audio analysis. Human hearing does not perceive all frequencies in the same way. The difference between 100Hz and 200Hz is not perceived the same way as the difference between 7000Hz and 7100Hz. Although the difference in both cases in 100Hz. The Mel scale (Stevens et al. [1937]) relates the measured frequency to the perceived frequency. It takes the original frequency outputs and remaps them as the human ear hears them. Many research teams (Hyeji and Jihwan [2019] Koutini et al. [2019] Chen et al. [2019]) have achieved state-of-the-art results using a Mel spectrogram type input feature. The Mel scale (Slaney [1998]) is defined as (5).

$$k = 2595 log_{10}(1 + \frac{f}{700}) \tag{5}$$

A derived input feature from the Mel scaled frequency transformation is the Mel frequency cepstrum (MFC). A cepstrum is a spectrum of a log Mel spectrum and captures the harmonic structures of sound. MFCCs are commonly used as features in speech recognition systems.

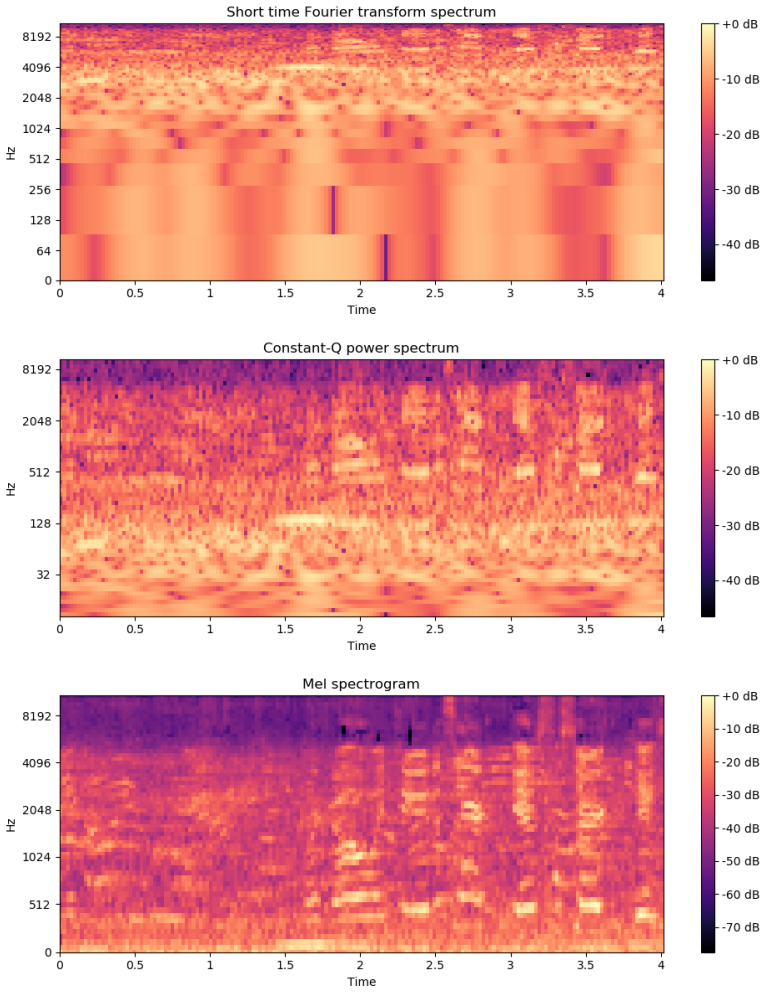

Figure 4: Comparison of the STFT, CQT and Mel spectrogram

Figure 4 illustrates the STFT, CQT and Mel transformation of the sound clip 100263-2-0-126.wav of the UrbanSound8k dataset. The audio belongs to the children playing class and is resampled to 22050Hz. The STFT shows a low resolution in the low frequencies when represented on a logarithmic scale. The CQT is converted using 6 bins per octave. The transformation shows more resolution in the lower frequencies compared to the STFT. The Mel transformation illustrates the Mel scale frequency warping where the 512Hz to 5kHz signal information in the signal is more spread out compared to the CQT spectrogram.

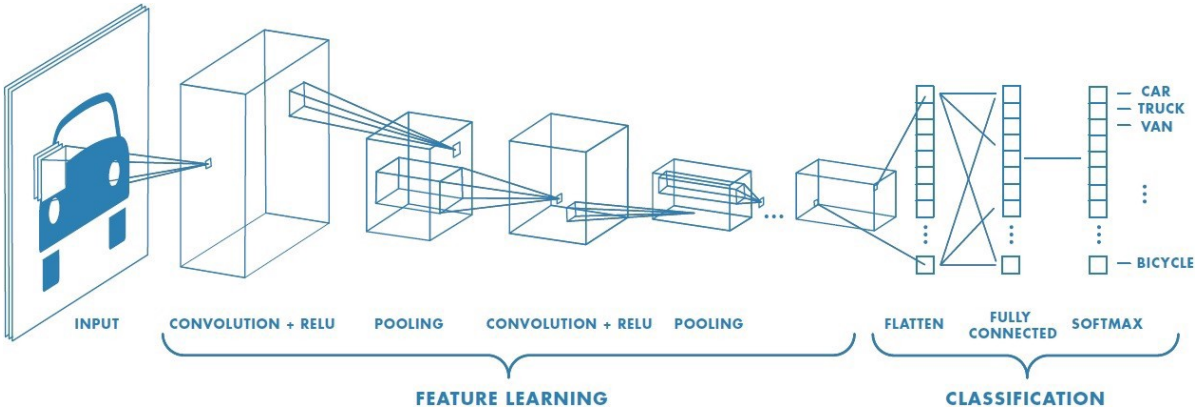

Figure 5: Convolutional neural netwerk architecture[1].

3. CONVOLUTIONAL NEURAL NETWORK

This section provides background information on convolution neural networks with more detailed sections for every layer of the network architecture. A convolutional neural network (Aloysius and Geetha [2017]) has the ability to extract features from input data via training. The approach is different from conventional machine learning classifiers, such as random forests (Ho [1995]) and support vector machines (Cristianini and Schölkopf [2002]), where the feature extraction is hand-crafted. Figure 5 illustrates the architecture of a vehicle classifier, made up of a feature learning part and a classification part. The feature learning part consists of one or more feature extraction blocks, each containing a convolution and pooling block. The first feature extraction block learns to recognize small basic elements in the input data. These can be lines, angles or curves, for example. The block scans the entire input image and detects where the basic elements occur. The information is passed on to the next feature block which in turn learns to combine these basic elements into visual aspects such as circles or rectangles. The detection of high-level features are assembled by the classifier block into a vehicle and divided into output categories such as car or truck. Once a feature has been learned, the feature can be detected at any position in the input data by the convolution algorithm. A classic classifier can also be taught to recognize a feature, for example at the bottom left of the input data, but should relearn the feature when it occurs at the top right of the input data. That is why convolutional neural networks are mainly used for image processing.

3.1. FEATURE LEARNING

Feature learning allows a neural network to discover patterns in the input data which can be used for classification. Features are extracted automatically during training in contrast to manual feature engineering (section 2.3.2). The feature learning block consists of two parts; the convolution layer and the pooling layer (section 3.1.3). The convolution layer is split up into classic convolution (section 3.1.1) and the mathematical optimized depthwise separable convolution (section 3.1.2).

[1]Source https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

3.1.1. CLASSIC CONVOLUTION LAYER

A convolution layer learns to recognize basic elements from the input data by using filters (Burkov [2019]). A filter is a matrix of numbers representing a specific pattern. Each filter generates a new image from the input data by convolving the filter with the input data. High values in the numeric convolution result indicate the locations where the filter pattern occurs the most in the input data. Figure 6 illustrates the result of horizontal and vertical edge filtering. The output image of the horizontal filter indicates positions where horizontal edges are detected but returns no indications for vertical lines and vice versa for the vertical filter. Both horizontal and vertical filters detect diagonal lines. Convolution is an operation that indicates to which degree there is an overlap between a function while it is being shifted over another function. The general formula is defined in (6).

Figure 6: Image filtering. Application of vertical line filter on the left, right horizonal line filter[2].

$$conv(I,F)_{x,y} = \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} \sum_{k=1}^{n_D} F_{i,j,k} I_{x+i-1,y+j-1,k} \tag{6}$$

The formula consists of a recursive multiply and accumulate (MACC) operation of the filter matrix F and the input data matrix I. The result is a sum of elementwise products where n_H,n_W and n_D represent the height, width and depth of the input data respectively. The number of MACCs indicates how computationally intensive a convolution operation is. The convolution is illustrated in figure 8 using a 3x3 filter matrix according to figure 7. The values of the filter, also called weights (white), remain constant during the convolution. The filter in this example represents an intersection of two diagonals. The filter is convolved with the input image (blue) of 5x5 pixels. The position of the 3x3 filter (dark blue) shifts in each step. The input data on which the filter is projected is called the receptive field of the filter. The convolution result is called a feature map. A feature map indicates where a particular feature occurs in the input data. If a convolution layer consists of multiple filters, the output will also consist of multiple transformed output images. The output is a collection of all two-dimensional feature maps making the shape three-dimensional. The filters of a subsequent convolution layer are therefore also three-dimensional. Due to this extra dimension, the number of operations increases. As a result, the convolution layers that follow after the first will also require significantly more processing power. There are two techniques to reduce processing power; striding and pooling (section 3.1.3).

Striding is a technique to downsample the input data. The stride is the step size of the moving window. In figure 8 the filter starts on the top left of the data and moves over all

[2] Source http://datahacker.rs/edge-detection/

$$F = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Figure 7: Filter matrix

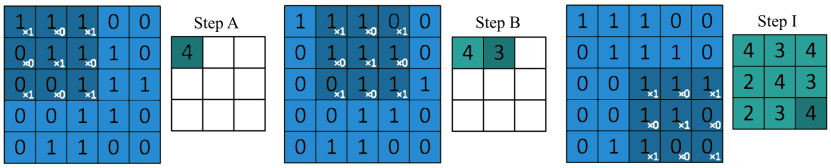

Figure 8: The blue matrix represents the input data with dark blue receptive field. The cyan matrix represents the output data.[3]

locations to the bottom right. Each time the filter moves one position, the stride is (1,1). When a pixel precise location of a feature is not required, the filter can slide faster and the stride step is increased. A (2,2) stride is illustrated in figure 9 which results in a smaller output map. When the stride step is more than 1, the size of the filter in combination with the stride step may not match the size of the input data. This means that data cannot be taken into account and is lost. This can be solved by data padding. Additional data is added to the input data so that no input data remains unused. The term no padding is used when the output dimension is smaller than the input dimension. Zero or same padding is used when the output dimension is the same as the input dimension.

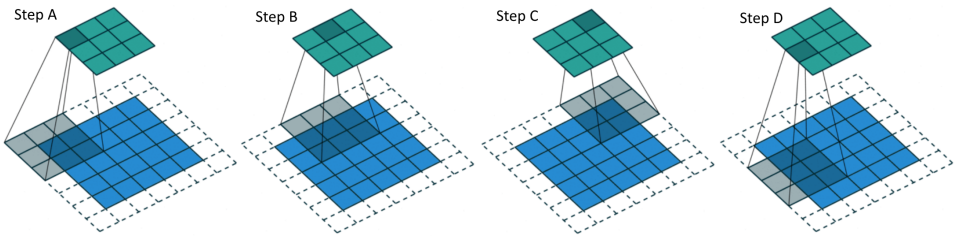

Figure 9: 2x2 Striding of padded 5x5 input data using a 3x3 filter. [3].

The output of the convolution is transformed by an activation function before the data is passed to the next layer. To learn complex relationships between input and output, a non-linear function is used as an activation function. In general, the hyperbolic tangent Tanh or Rectified linear unit (Relu) function are used (Géron [2019]). The Tanh has the advantage that all values are mapped between 1 and -1, centered around 0. The disadvantage is vanishing gradient, which means that for very large positive or negative values, there is almost no difference in function value anymore. The Tanh function requires a lot of computational power. The Relu function is computationally more straightforward and therefore more efficient.

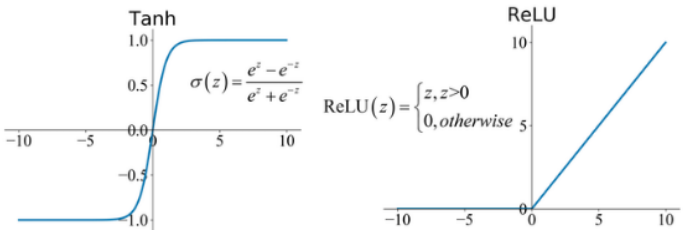

Figure 10: Activation functions Tanh (left), Relu (right). Feng et al. [2019].

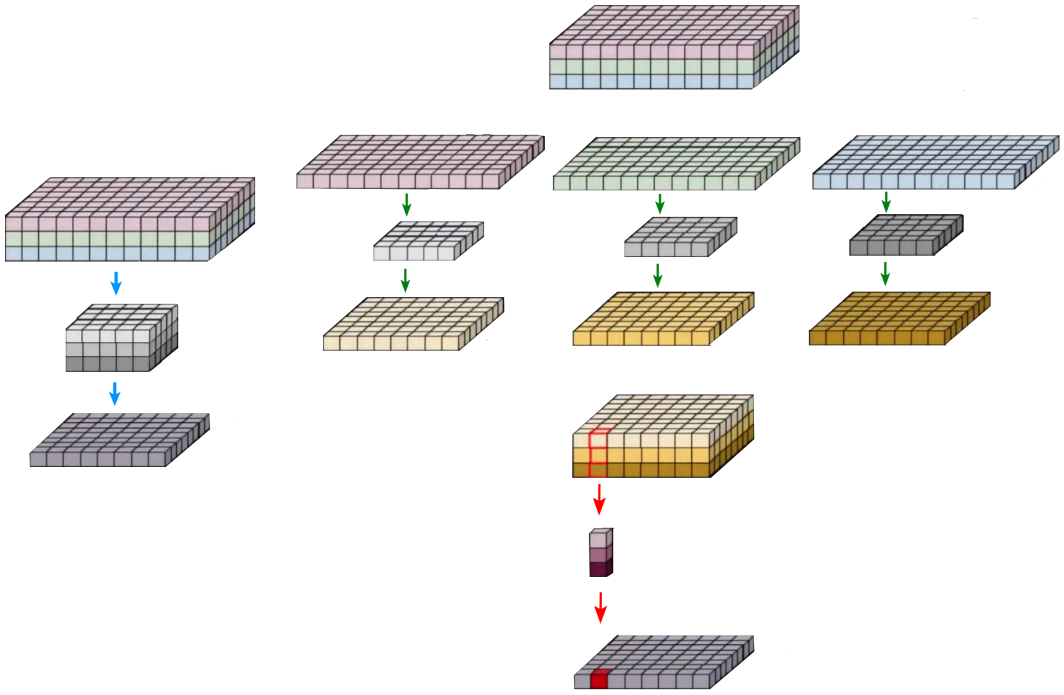

Figure 11: Classic convolution (left) versus depthwise convolution (right) using 5x5 filter on 12x12x3 input.

3.1.2. Depthwise separable convolution layer

Depthwise separable convolution (Chollet [2016]) is the mathematically optimized version of classic convolution. It is an important technique to reduce the required computational power in a convolutional layer by reducing the number of multiplications. On a hardware level, the multiply operation requires a higher energy budget compared to the add operation (Vasilyev [2015]). The optimization consists of splitting the convolution into two parts; depthwise convolution and pointwise convolution. Figure 11 illustrates the default convolution on the left using the blue arrows, as discussed in Section 3.1.1. The depthwise separable convolution is illustrated on the right with the depthwise convolution using green arrows, followed by pointwise convolution using red arrows.

In classical convolution (Section 3.1.1), each filter is multiplied by the input data while the filter is shifted over the input dimensions. For example, a 5x5x3 (k_Hxk_WxM) filter with a 12x12x3 (HxWxM) input image produces an 8x8 (F_HxF_W) feature map. This requires 5x5x3 or 75 multiplications on each of the 8x8 positions resulting in 75x64 or 4800 multiplications for the convolution of one filter. When 256 (N) filters exist in a model, this results in 75x64x256 or 1228800 multiplications. The generic number of operations in a classic convolution can be given according to (7). If the height and width are the same for the filter and input image, the number of operations is defined by (8).

$$O_{\text{Conv}} = Nk_H k_W M F_H F_W \tag{7}$$

$$O_{\text{Conv}} = NMk^2F^2 \tag{8}$$

In depthwise separable convolution, the same input image of 12x12x3 (HxWxM) is taken in the depthwise convolution part, but instead of a 5x5x3 filter, three separate filters without depth of 5x5x1 are used. Each layer of the input data is then convolved with one filter

[3] Source https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

resulting in three times an 8x8x1 feature map (green arrows). At that point, there is no relationship whatsoever between the different layers because each layer is processed separately along the depth direction. The red, green and blue layers are not mixed. The number of multiplications is then 5x5 for the filter applied to 8x8 positions or 5x5x8x8 or 1600 for each input layer. For the three input layers together, this is 4800 multiplications. The generic number of operations is defined by (9) or if the height and width are the same for the filter and input image by (10).

$$O_{\text{Depthwise}} = k_H k_W M F_H F_W \tag{9}$$

$$O_{\text{Depthwise}} = M k^2 F^2 \tag{10}$$

The pointwise convolution part ensures that the end result has the same output shape as the classic convolution. A 1x1x3 (1x1xM) filter is used on the combination of feature maps. The filter has a 1x1 area of 1 point or pixel but has depth (M) of all layers. Hence the filter has no spatial relationship but a depth relationship. The 1x1x3 filter moves 64 times on the 8x8 (F_HxF_W) output feature map area (red arrows), combining all feature maps. The number of multiplications is then 3 per filter applied at 8x8 positions or 3x8x8 = 192. When 256 (N) output features maps are required, the number of operations is 192x256 = 49152. The total number of multiplications required for the depthwise separable convolution is the sum of 4800 and 49152, which results in 53952 multiplications. The generic number of operations according to (11) or if the height and width are the same for the filter and input image follows (12).

$$O_{\text{Pointwise}} = N M F_H F_W \tag{11}$$

$$O_{\text{Pointwise}} = N M F^2 \tag{12}$$

The total number of multiplications for depthwise separable convolution is the sum of the depthwise and pointwise convolution steps. If the height and width are the same for the filter and input image, the total number is according to (13).

$$O_{DS} = O_{\text{Depthwise}} + O_{\text{Pointwise}} = M k^2 F^2 + N M F^2 = M F^2 (k^2 + N) \tag{13}$$

The ratio between the number of multiplications for depthwise separable convolution and the classic convolution is then according to (14).

$$\frac{O_{DS}}{O_{\text{Conv}}} = \frac{M F^2 (k^2 + N)}{N M k^2 F^2} = \frac{k^2 + N}{k^2 N} = \frac{1}{N} + \frac{1}{k^2} \tag{14}$$

The multiplication ratio of a network with 256 filters with a size of 5x5 is 0.044. The depthwise separable convolution therefore requires 22.78 times less multiplications than the classic convolution. Classic convolution transforms the input image 256 times at 4800 multiplications per transformation. While with depthwise separable convolution, this only happens once in the depthwise part and where this data serves as input 256 times in the pointwise part. The decrease in multiplications directly impacts the time to calculate the math graph or inference time. In addition, the depthwise separable convolution also has an impact on the number of parameters that are trained.

The number of learnable parameters for a classic convolution is the number of parameters in the filter times the number of filters or NxMxk^2 according to (15). The number of learnable parameters in depthwise separable convolution is the sum of the number of parameters in the depthwise and pointwise convolution. In depthwise convolution, the

number of parameters is the parameters in the filter times the depth of the input data or Mxk^2 according to (16). In pointwise convolution, the number of parameters is 1x1xM parameters times the number of filters or MxN to be trained according to (17).

$$P_{\text{Conv}} = NMk^2 \quad (15) \qquad P_{\text{Depthwise}} = Mk^2 \quad (16) \qquad P_{\text{Pointwise}} = NM \quad (17)$$

$$\frac{P_{DS}}{P_{\text{Conv}}} = \frac{Mk^2 + NM}{NMk^2} = \frac{M(k^2+N)}{NMk^2} = \frac{k^2+N}{k^2N} = \frac{1}{N} + \frac{1}{k^2} \quad (18)$$

The ratio of the number of learnable parameters between depthwise separable convolution and the classical convolution according to (18) is the same as the ratio of the number of multiplications between depthwise separable convolution and the classical convolution according to (14). As a result, the number of learnable parameters will drop by the same ratio and training time for depthwise separable convolution models will be shorter. There is a trade-off between the number of model parameters and the model flexibility. The model will be unable to capture the underlying pattern of the data if it has too few parameters.

3.1.3. Pooling layer

Pooling layers are besides striding another way to reduce computation power across the network (Géron [2019]). Then input of a pooling layer is the output of a previous convolution layer. During the pooling, a filter is shifted over the input data. This filter has no weights but performs an average or maximum function. Figure 12 illustrates a 3x3 max pooling filter with the input in the green frame and the output in the purple frame. A pooling layer is not only beneficial for computational power. Without the pooling layer, a small move of the position of a feature in the input would result in a change of the feature map. With the pooling function, this can lead to the same feature map. This property is called the translation invariance of a model. In concrete terms, this means that when the input shifts by a small step, the output is hardly changed.

The pooling layer is the last operation of the feature extraction block. The output can be processed by another feature extraction block that extracts higher level features from the features maps of the previous block. When sufficient features are extracted, the information is passed onto the classification block via the flatten layer.

3.2. Classification

The classification part of the convolutional neural network is responsible to map the extracted features to a set of output classes. It consists of a flatten layer, a fully connected network and an output layer.

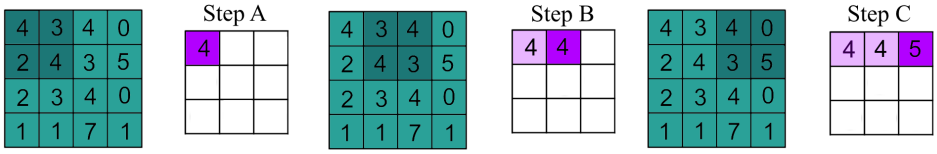

Figure 12: Maximum pooling of the 4x4 convolution output results in 3x3 output.

3.2.1. FLATTEN LAYER

The flatten layer is the link between the feature extraction block and the classification block from the model architecture in figure 5. The role of the flatten layer is to reshape the 3D convolutional tensor data without changing the data content. The flatten layer flattens all data and removes all but one dimension.

3.2.2. FULLY CONNECTED NETWORK

A fully connected network (Burkov [2019]) is the central part of the classification from the convolutional neural network architecture in figure 5. A fully connected network or dense network consists of a series of fully connected layers with neurons where each neuron in one layer is connected to all neurons of the next layer. Figure 13 illustrates a fully connected network with four layers. The first and last layers are respectively called input and output layer. All inner layers are called hidden layers. Each neuron or perceptron is a computational block that triggers its output when enough stimuli are present on the inputs according to figure 14. Each neuron in the input layer takes in a scalar value. The input data comes from the flatten layer that converted the extracted feature data into a 1-dimensional vector.

$$\text{output} = \text{actFunc}\left(\sum_{i=1}^{n} x_i w_i + \text{bias}\right) \tag{19}$$

A neuron multiplies each input by a dedicated weight. The sum of all products is converted by an activation function according to (19). The layers are executed sequentially during classification. All data is applied synchronously to the input layer. Then the output of all neurons is calculated and used as input for the next layer. Next, the neurons of the next layer are calculated, etc. The sequence of calculations is called forward propagation.

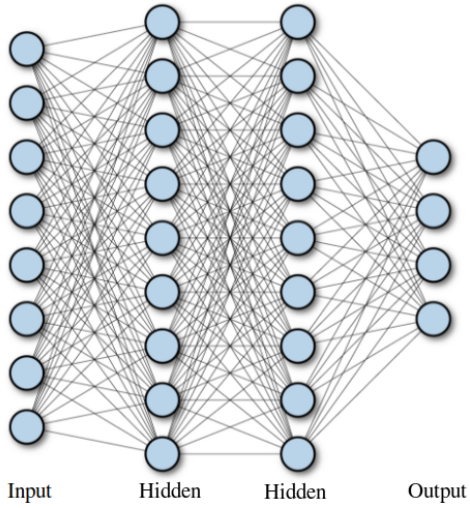

Figure 13: Fully connected network

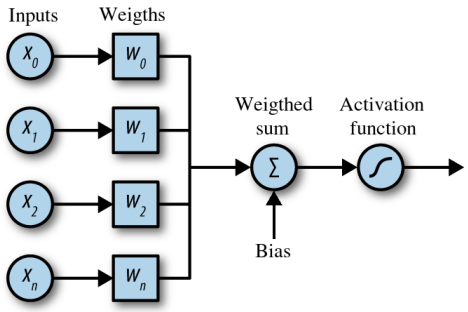

Figure 14: Perceptron [4]

3.2.3. SOFTMAX LAYER

The output of the classifier is typically a softmax layer. The softmax layer takes the set of output values from a fully connected layer as input vector and converts the values into real numbers whose sum is 1. The values can be negative, positive or greater than 1. After the

[4] Source: https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html

softmax function, the values are between 0 and 1 so that they can be treated as probabilities. The softmax function is defined according to (20) where z is the input vector and C the number of output classes.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{20}$$

3.3. Training process

The training process of a neural network can be divided into three steps; the forward pass, loss function and backward pass. During the forward pass, input data is presented to the model and the algorithms of all consecutive layers are executed based on the current parameters. The output data is used by a loss function to evaluate how far the estimated classification is from the true classification. The backward pass estimates the amount of error for which the weights of neurons are responsible and updates the weights accordingly.

Generally, the training process starts with randomly initialized weights. When a subset of the training data has been processed, the weights are updated to reduce the error. The amount of change applied during each iteration is known as the learning rate. The processing of a complete training dataset is called an epoch. After a number of epochs, the training process fits the model to the training data. The model accuracy is evaluated on data the model did not see during training. Therefore a part of the dataset is hold out for testing. A model is underfitted when it is unable to predict the labels of the training data well. Reasons for this might be the model architecture can be too simple or the data features do not contain sufficient information. A model is overfitted when the model predicts the training data very well but poorly predicts the test data. The model can be too complex for the data or not enough training data might be available.

A trained model can also be the start point of a new training. This method is known as transfer learning. The idea is to improve learning a new task by reusing knowledge learned on a related task. In context of convolutional neural networks, the learned filter weights can be the starting point for training instead of random initialized values.

A training process involves intense matrix multiplications of the network model on large datasets which can take several days. Graphical processing units (GPU) can significantly accelerate the training process due to their massive parallel computing architecture. In traditional machine learning solutions, the network model is trained and deployed on GPU-enabled machines. In this research, the embedded system imposes computational constraints hence the network models are trained on a different platform than where the model is deployed.

3.4. Related work

In the past years, much research has been performed on optimizing CNNs (section 3.1.1) for mobile devices (Alippi et al. [2018]). MobileNet (Howard et al. [2017a]) is one of the first architectures designed for embedded systems with an efficient trade-off between model latency and model accuracy. It was one of the first CNNs that used depth separable convolution to decrease the number of computations. It refactors the standard convolution into a depth convolution and a pointwise convolution (section 3.1.2). During the depth convolution, each input channel is convolved with a separate filter. During pointwise convolution, the output is combined using a 1x1 kernel convolution. This significantly reduces

the convolution computation.

Zhang et al. [2017] have designed a model called ShuffleNet. Two new operations have reduced the computational effort; pointwise group convolutional and channel shuffle. The group convolution used in Alexnet (Krizhevsky et al. [2012]) did not allow information flow between channel groups. The accuracy of Shufflenet is 7.8% better than MobileNet. The model is 13x faster than Alexnet with comparable accuracy. The efficiency of the structure.

Piczak [2015b] was one of the first optimized CNN models for ASC. The shallow design contains two convolutional layers with max-pooling followed by three fully connected layers. The NN has two input features; a log-Mel spectrogram and the first-order difference (or sample delta). Each convolution layer has 80 kernels. The kernel sizes are defined as 57x6 for the first layer and 1x3 for the second layer. The solution was validated on three public datasets; ESC-50, ESC-10, and Urbansound8k. An accuracy of 69% to 73% was achieved using implementation varying short/long segment analysis and majority/probability voting.

Salamon and Bello [2016] have improved the Piczak [2015b] accuracy on the Urbansound8k dataset with a shallow CNN called SB-CNN. The design consists of three convolutional layers with max-pooling and two fully connected layers. The convolution layers have 24 filters for the first layer and 48 filters for the second and third layers. The kernel size is 5x5 over all layers. The kernel sizes are defined as 57x6 for the first layer and 1x3 for the second layer. The model accuracy of 73% is comparable with Piczak. However, accuracy reaches 79% using data augmentation. With data augmentation, the limited dataset is extended by transforming the audio samples and creating new extra samples for the network. This can be achieved by applying for example time stretching or pitch shifting the audio signal, similar to rotating or skewing images for an image recognition task.

Nordby [2019] has optimized the SB-CNN model further. The max-pooling is 3x2 instead of 4x2 and batch normalization was added to each convolutional layer for the baseline model. A strided model is also presented with removed max-pooling in favor of increasing the striding of the filters of the previous convolution layer. When the convolution layers are implemented using depthwise separable convolutions, a mean accuracy of 70.9% was reached on the Stride-DS-24 model using data augmentation. The smallest required parameter size is 55kB RAM and 96kB flash for the Baseline-DS model (477kMAC).

Nossier et al. [2019] present a smart hearing aid enhancement using deep neural networks. The feature tries to distinguish between desired and undesired noise types. The work presents three solutions to detected important noise like a fire alarm or car horn to and makes it audible to the impaired while discarding other noise. The audio speech samples were used from the Voicebank dataset, the noise samples were from the Urbansound8k dataset and ESC-50 dataset. The CNN used to classify the sounds was based on Piczak [2015b]. The model uses two channels of Mel spectrograms (plain and delta). The accuracy of the original Piczak [2015b] model on the Urbansound8k dataset ranged from 69-73%. The dropout was reduced from 50% to 20% in this research. The research does not mention model deployment nor provides details of any hardware used.

Although the interweaving design of Sinha and Ajmera [2018] claims to achieve higher accuracy on the Urbansound8k dataset than SB-CNN. The model is not optimized for embedded devices.

Previous research has demonstrated the successful application of large convolutional neural network models in context of audio event classification. Although the models can run on mobile devices like smartphones, the required computational power and memory size are still significant. Recent work of Nordby [2019] focused on model efficiency where the SB-CNN model of Salamon and Bello [2016] was optimized for deployment on low-end microcontrollers. However, the reported model sizes are too large to fit the 128kB memory limit of tiny embedded devices.

4. QUANTIZATION

This section provides background information on quantization in the context of convolutional neural networks. Quantization refers to conversion techniques that optimizes neural networks so that they can be used on low end embedded devices. A neural network can be optimized by reducing its arithmetic precision. A quantized model performs its tensor operations with integers rather than floating point values. The quantization approach in this research reduces the 32bit floating point parameters to 8bit integer values. Representing numbers with this precision requires 1 byte of memory space for each network parameter instead of 4 bytes. The memory storage requirements are reduced by a factor of 4. Because less bytes are stored in memory, less bytes need to be transferred from memory to the CPU. The lower memory bandwidth results in lower energy consumption. Integer math is less complex than floating point math. Therefore integer operations are faster than floating point operations. The computational processing time required to perform an inference on neural network will be shorter. This is an advantage for the power constraints where the battery life can be extended by reducing the awake time. On the other hand, more complex models can be deployed within the same computational resources.

Another advantage is the portability of an integer based algorithm. Only high end embedded microcontrollers have hardware support for floating point calculations, but all embedded devices support hardware integer operations (section 2.1). Some architectures have hardware acceleration for integer operations. The single instruction multiple data or SIMD instruction accelerates integer operations on ARM architectures. It does not require 8bit values to be stored into separate CPU registers before an operation. Instead, the instruction allows 8bit integers to be grouped together into one 32bit register and can apply the same operation on all 8bit integers simultaneously. This significantly improves the performance.

The disadvantage of quantization is that the parameters can not always be represented without precision loss. The rounding of the numbers can lead to a drop in the network accuracy. However, convolutional neural networks are by nature robust against noise (Warden and Situnayake [2019]). Even when the numbers used are rounded, the result is reasonably accurate. This is because the input data is full of noise. For image recognition, this is the noise from the camera chip or shadow elements in the image. Models learn during their training to distinguish these insignificant differences from the real features. Therefore, the 32bit precision is typically too precise for the calculations and the numerical precision can be optimized.

In general, quantization is the representation of values from a continuous range to values in a limited range. In concrete terms, the 32bit floating point numbers are downscaled and represented by 8bit integer numbers. The reason why quantization is possible on convolutional neural networks is because the range of the filter weights is relatively small. Figure 15 illustrates a histogram of all filter weights from the SB-CNN model (section 5.2.2). In this example, the values are between -4.451389 and 6.2823944. Almost all weights are relatively close to zero, so there is no need for a large 32bit floating point range to represent the weights. During the quantization in figure 16, the values 0 to 255 from the 8bit range are mapped linearly between -4.451389 and 6.2823944. Each floating point number is therefore pushed into one of the 256 boxes. As a result, 256 floating point numbers from the range will get an exact conversion, all others will undergo rounding. The pseudo-code for the quantization:

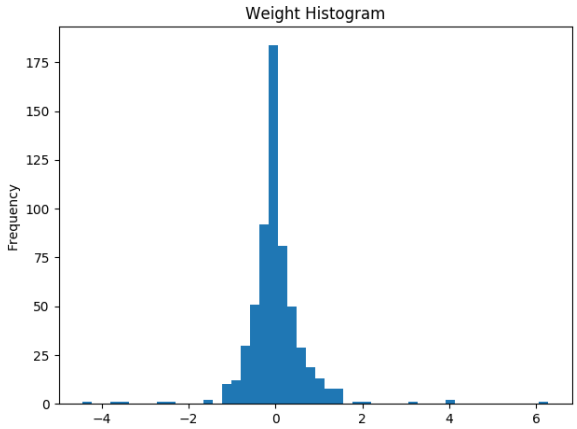

Figure 15: Histogram of kernel weight values of first convolution layer of SB-CNN model.

$$\text{quantizedValue} = (\text{uint8})(\frac{\text{originalValue} - \min}{\max - \min} * 255) \tag{21}$$

$$\text{reconstructedValue} = (\frac{\text{quantizedValue}}{255.0} * (\max - \min)) + \min \tag{22}$$

Figure 16 is an example of asymmetric quantization because the absolute values of minimum and maximum are different. The representation of the values is not centered exactly around zero. The disadvantage is that the value zero requires rounding. As the histogram in figure 15 indicates, zero is the most common value in kernel weight distribution of the convolutional layer. If the floating point 0 is not equal to the integer 0, an encoding error is introduced. The error will be repeated much more for the number 0 compared to the other values. This can lead to a bias throughout the network, which will decrease accuracy. The value of 0 is also used for padding around the edges of images in convolution layers (section 3.1.1).

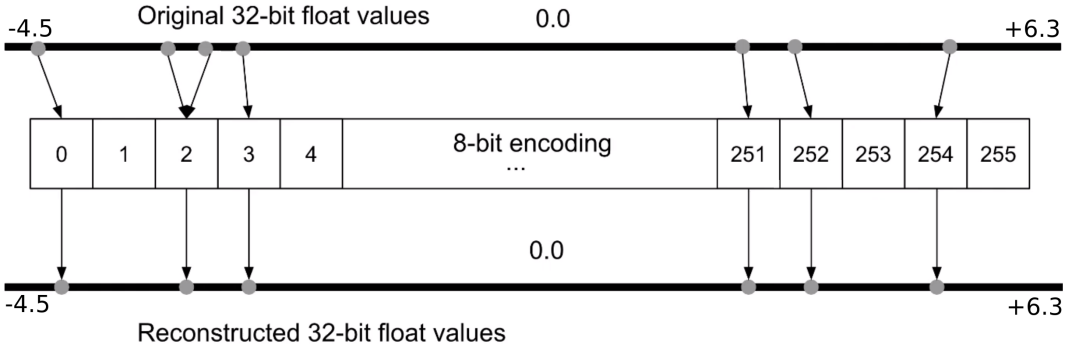

Figure 16: Asymmetric qantization range [5]

Figure 17 is an example of symmetric quantization because the absolute values of minimum and maximum are equal. All values are centered around zero on the floating point side. One integer value has been omitted to make this possible. Signed integer values have

a number more on the negative side than for the positive numbers due to the two complement representation. Therefore the number -128 has been omitted to make the floating point 0 value exactly match the integer 0 value. The disadvantage of the symmetric encoding is that a part of the scaling range is lost. The largest absolute value of the weight distribution is used to encode all possible weight values. In the example of figure 17, the range is set to -6.3 to +6.3, while there are no weight values below -4.451389 in the histogram.

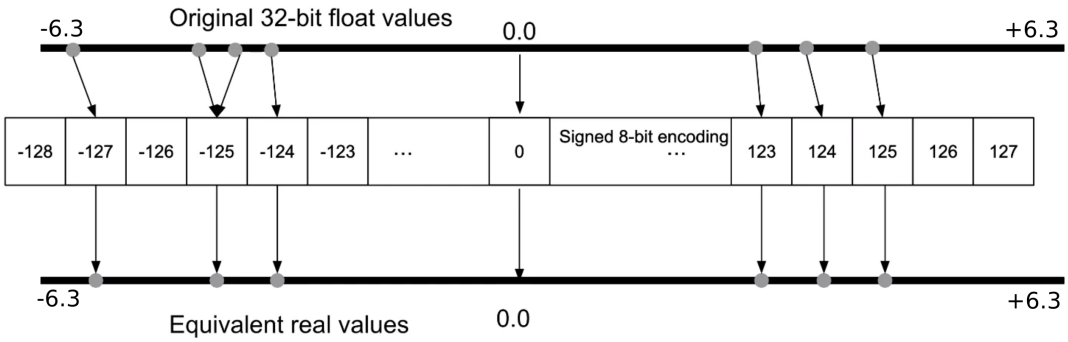

Figure 17: Symmetric qantization range

Figure 18 illustrates a 5x5 filter from a convolution layer. The 25 floating point weights are limited to 4 significant numbers for clarity of the example. The shade of gray of each box is proportional to the value of the weight. The maximum value corresponds to white, the minimum value corresponds to black. Figure 19 shows the quantized version of the same 5x5 filter. Each box contains two numbers. The top number is the quantized integer value calculated according to (21). The bottom number is the reconstructed value in floating point precision calculated according to (22). The white area represents the maximum number 6.2824 by +127. The maximum number has no rounding error. All other numbers do have a visible rounding in their reconstructed value. The grayscale pattern is similar but not the same.

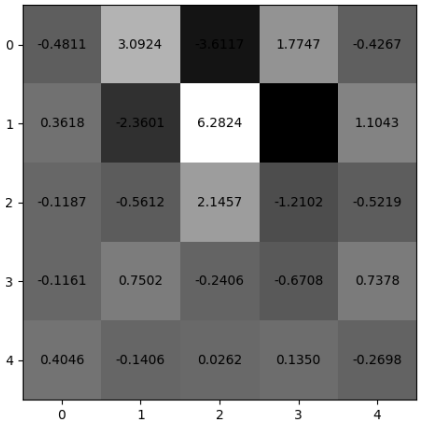

Figure 18: Floating point 5x5 filter SB-CNN.

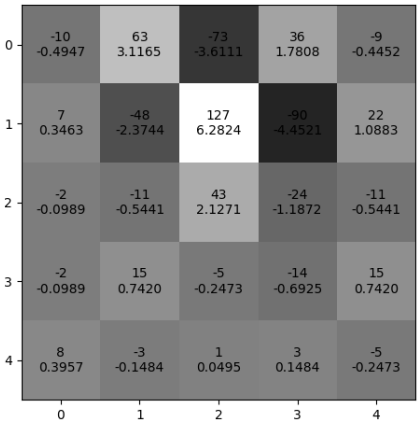

Figure 19: Quantizated 8bit integer 5x5 filter SB-CNN.

The two main techniques for quantizing neural networks are post quantization and training aware quantization. Post quantization starts from a trained floating point model in

[5] Source: Screencast Stanford university course EE282D https://www.youtube.com/watch?v=-jBmqY_aFwE

which all weights are converted to a lower precision. To find an efficient scaling and offset, the process uses example data. During the quantization, it is examined how typical input data relates in the network. Based on those values, the dynamic range is set. Another technique is quantization aware training and is already applied during the training. By using false nodes in the network, the process tries to direct the weights to values close to the final quantization values. Both techniques are explained in more detail in the next sections.

4.1. POST QUANTIZATION

Post quantization (Krishnamoorthi [2018]) is a conversion technique that is applied after a model has been fully trained in floating point. Figure 20 illustrates quantization in a basic step of a filter convolution (section 3.1.1) in which a dot product is made with the input data and the weights from a filter. The color spectrum indicates the range of a floating point number. During quantization, a linear transformation is made from a limited color area within the spectrum to an 8bit integer. When an 8bit weight (step A) and 8bit input (step B) are multiplied, a maximum 16bit result is obtained. The products of individual inputs and their filter weight are summed to a dot product. The addition of a 16bit number with a 16bit number gives a maximum of a 17bit number. Depending on the filter size, this becomes a number with a larger range according to $log2(m.2^n) + 1$ (step C) for m n-bit numbers. As with the inputs and weights, this is in a certain range that has to be requantized to 8bit to go to the next operation (step D).

The quantization of the weights is relatively easy because these are constants. The scaling and offset can be calculated because the minimum and maximum value of all weights of layer are known. This is more difficult for the activations. Because the activation input depends on the dot product of the input and the weights, it is not known in advance what the possible minimum and maximum range will be. By taking the theoretical minimum and maximum for the inputs, the output range of the dot products becomes very large compared to the results with real input values. A large part of the quantization range would not used. This ensures that due to the quantization the values almost all end up in the middle box (see figure 17). Due to rounding, there is an insufficient distinction between the values, which is detrimental to the accuracy. That is why the activation functions are calibrated by running real data through the network during the post quantization process. The minimum and maximum values can be precisely recorded with real data, resulting in better scaling.

The disadvantage of post quantization is that an existing model cannot simply be quantized without a real dataset. During the deployment phase, information is needed from the training phase, which is not always available.

4.2. TRAINING AWARE QUANTIZATION

Training aware quantization (Jacob et al. [2017]) is a technique in which the training of a model takes into account the lower precision of quantization. Figure 21 depicts the graph of a normal convolutional flow on the left versus the training aware quantization graph on the right where two fake quantization nodes are added to the graph. During the forward pass, the floating point weights are converted into quantized weights before they are used. The quantized weights are convolved with the input data after which the bias is added. After the

[6] Source: https://sahnimanas.github.io/post/quantization-in-tflite/

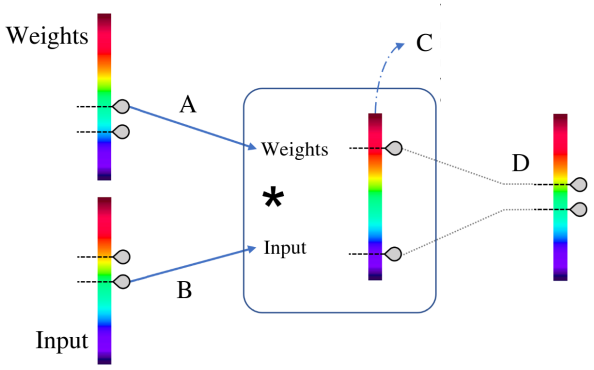

Figure 20: Quantization of inputs, weigths and their dot product[6]

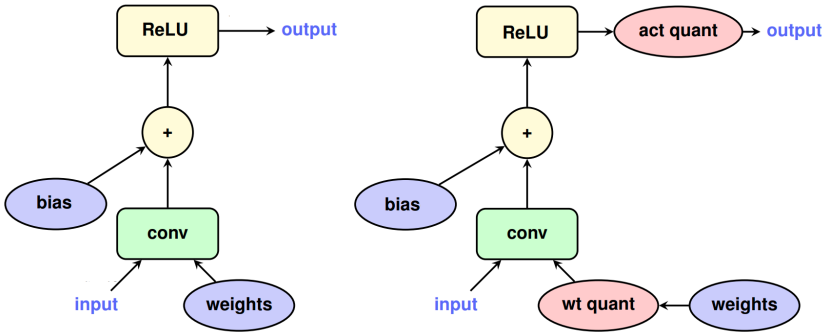

Figure 21: Left basic flow convolution layer graph, right training aware quantization graph (Source Jacob et al. [2017])

activation function, the output is first converted into a quantized value before the value is passed on to a subsequent operation. The backward pass is done in the normal way with the floating point values of the weights being adjusted with very small gradient updates. By injecting the rounding errors that occur during quantization, the network can be better tuned to the rounding errors. This technique does not require a calibration dataset because the minimum and maximum ranges are maintained by the fake quantization nodes. The disadvantage is that existing models have to be retrained and the extra steps in the training process make the training time longer.

4.3. Quantization frameworks

A quantization framework is an ecosystem of tools and libraries that bring neural network models to mobile, embedded and IoT devices. The development flow consists of multiple stages; select a model, retrain the model (optional), optimize the model by quantization and deploy the model onto a device. One of the most popular quantization frameworks is TensorFlow Lite. Tensorflow is an open source platform for machine learning with a high-level application interface called Keras to design, train and evaluate network models. The Tensorflow Lite extension is used to optimize and deploy models on microcontrollers. The platform offers many pre-trained models from image, speech or gesture recognition to text, digit or sound classification. Any existing floating point model can be used as a starting point. In case of post quantization, the model can be converted from 32bit floating point to 16bit floating point if the hardware target has 16bit floating point support. The memory

budget is divided by 2. If a representative dataset is available, the dynamic range of the data flowing through the network can be measured (section 4.1). The quantization scaling from 32bit floating point to 8bit integer is calculated per filter to optimize the rounding for each data channel. In this case the memory budget is divided by 4. In case of training aware quantization, the model is retrained before the quantization step. At optimization stage, the model is still in a Keras format, containing all trained parameters, layer architecture, training information, etc.. In the deployment stage, the model is stripped from all metadata. The trained parameters are converted into a flat buffer format or a raw collection of data in a specific format. At this point the interpreter comes in. The interpreter is the core engine which can load the trained parameters and actually execute the model algorithm. The interpreter runs on the target platform and is optimized for the hardware architecture. The TensorFlow Lite for Microcontrollers is a tool that generates the source code for multiple embedded architectures like ARM Cortex, ESP or ARC. The interpreter is written in C++ and requires no standard C or C++ libraries. Another tool for deployment is the X-Cube-AI tool by microcontroller manufacturer STM32. This tool can load keras or tensorflow lite models and convert them to C code for their STM32 microcontroller families. The inference engine is generated by high optimized C code which allows the model to run on the smallest architectures like ARM Cortex-M0. The tools are deeply integrated with the embedded devices and allow precise performance measurements using onboard hardware timers.

4.4. Related work

This section describes related research on quantization in context of convolutional neural networks. The proposed solutions have driven semiconductor chip designs.

[Lin et al., 2015] propose a quantization design that identifies the optimal bit-width allocation for each individual layer. A forward pass in floating point is executed on typical input data set to collect the statistical data of all layer parameters. Next, the ideal bit-width fix point formats are determined for each layer. The effect of the quantization can be defined as the signal to quantization noise ratio (SQNR). A higher quantization noise leads to lower accuracy. [Lin et al., 2015] calculated the SQNR throughout a complex CNN structure and concluded that the total SQNR is the harmonic mean of the SQNRs of all individual quantization steps. This means that all quantization steps have the same weight on the total SQNR and the worst quantization step will dominate the outcome.

Lai et al. [2017] show that floating-point representations for weights are more efficient than fix point representation for the same bit width. The requirements request three properties. The network must achieve sufficient accuracy with limited bit-width. The method should work consistently for different CNNs and it should have an efficient way to implement it in hardware. The proposed solution shows that the multiplication of one floating point operand and one fixed point operand can be more efficient than two fixed-point operands. A floating-point number consists of a sign bit, mantissa, and exponent. When converting a real number to floating-point number, the integer part before the radix point is converted to binary format and the fractional part is converted to binary format. Next, the radix is moved until the number is completely fractional. At every position move, the number is divided by two. To keep the value correct, the number is multiplied by 2 to the

power of the number of radix point movements. This is the exponent part. The idea of Lai et al. is that the mantissa can be multiplied with the fixed point operand which will result in a fixed point operand. Next, the exponent will be implemented in hardware by binary shifts. This is faster than a higher bit number multiplication.

Choukroun et al. [2019] proposed a low bit precision linear quantization framework that converts existing pretrained NN models to run on hardware targets without floating-point precision. The method minimizes the noise power of the weights and activation functions via mean squared error analysis to approximate the original model accuracy. The bit width could go down to 4 bit for popular neural network architectures.

The state-of-the-art quantization research lowers the numeric precision of 32-bit floating-point models down to 4-bit integer precision for both weight and activations. Choukroun et al. [2019] evaluated their quantization framework on non-over-parameterized networks, which are usually not analyzed in NN literature. The loss in accuracy for SqueezeNet ranges from 5.9%-7.4 and 4.4%-7.6% for Densenet. This allows low-bit inference without the need for full retraining. The granularity of the quantization depends on the hardware architecture of the target platform.

The overview of Krishnamoorthi [2018] indicates accuracies within 2% for a variety of CNNs for per-channel quantization and per-layer quantization in 8bit post-training quantization. They demonstrated accuracy can be improved using training aware quantization within 1% of floating-point using 8bit precision and 2%-10% when using 4-bit precision. Network models with fewer parameters like Mobilenet are less robust to quantization than models with more parameters. The compared models MobileNet, NasMobile, Inseption, and Resnet variants have a model size in the order of MBs.

Jacob et al. [2017] found out that the post-quantization approach works well on large network models with considerable representational capacity but would lead to a significant accuracy drop on smaller models. The failure modes are assigned to two root causes. The first is the large difference in ranges of weights for different output channels. The second cause is the outlier weight values which influence the scaling (Figure 15) and makes the remaining weights less precise after the quantization.

The experiments of Sheng et al. [2018] have demonstrated that not all existing network designs are suitable for quantization. The lightweight MobileNetV1 indicates a large accuracy gap after quantization compared to the over-parameterized VGGNet, GoogleNet or Resnet. They propose a new quantization friendly separable convolution method for MobileNets.

Many of the proposed solutions uses dedicated hardware multipliers which can not implemented efficiently in firmware. The work will drive new microcontroller chip designs. Unfortunately, embedded systems of today have non-configurable hardware multipliers with fixed bit-witdh length. Therefore the bit-width is fixed to 8, 16 or 32bit. Although 8bit quantization has been successfully applied to large well parameterized neural networks, research work also indicates that quantization might not work well on small models.

5. RESEARCH DESIGN

Convolutional neural networks achieve near-human performance on audio classification tasks. The research in this area used to be primarily on gaining the highest accuracy regardless of the required computational power. In recent years, more research was published in optimizing neural networks. This allows CNNs to be deployed on low-end IoT devices. These low-end embedded devices are built on battery-powered microcontrollers. Implantable medical devices have even more hardware constraints. In order to gain insight into how efficient audio classification can be performed on implantable hearing devices, some challenges must be considered.

The first challenge is related to hardware resource constraints. Implanted medical devices are tiny devices that are battery powered. The battery life constrains the available memory, processing power and processor complexity. Tiny microcontrollers have no support for energy efficient floating point arithmetic. Many of the shelf models are floating point models. Quantization is an optimization technique which can reduce the arithmetic precision from floating point numbers to integers. Integers are smaller which is beneficial for the memory and can be executed faster. Although quantization has been successfully applied to well parameterized neural networks, research work also indicates that quantization might not work well on small models. Jacob et al. [2017] found out that the post-quantization approach works well on large network models with considerable representational capacity but would lead to a significant accuracy drop on smaller models. The highest reported audio classification performance on low-end microcontrollers using small floating point CNN models was hypothesized by Nordby [2019]. The research evaluated models with classic and optimized depthwise separable convolution on the UrbanSound8k dataset. The work of Nordby is used as baseline for this research.

The second challenge is associated with the nature of the sounds that need to be classified. A person could be listening to music, having a conversation at home, or could be walking through a crowded street. Depending on the acoustic scene, the hearing device optimizes its acoustic settings to interact better with the environment. This task is referred as sound scene classification. The external sounds can be interfered with the body sounds picked up by the subcutaneous microphone. The sound of hair scratching, a mouth mask rubbing the skin or the sound of water running over the skin are much shorter. These sounds are detected with sound event classification. Consequently, the neural network model must be capable of classifying both types of sound.

The third challenge is linked to the microphone signal spectrum. A cochlear implant is powered by an external sound processor worn on the ear. The external microphone captures sound air waves from the environment which are communicated wirelessly to the implant to stimulate the auditory nerve. When the sound processor is removed, the cochlear implant is powered by an internal battery and switches to a subcutaneous microphone. The construction of an implantable microphone is different compared to a classic microphone hence the frequency sensitivity is different (Calero et al. [2018]). The implanted device uses two different microphone signals. When the sound processor is used to recharge the implanted battery, the sound signal source switches from the subcutaneous microphone to the signal provided by the external sound processor microphone. The choice of data feature is important as the spectral content is different. The Mel transformation and constant Q transformation dominate audio classification research (Song et al. [2019]). Although the features have been compared by Huzaifah [2017], the investigation was performed on large

models and did not take into account different microphone responses.

5.1. Research questions

The main research question is "To which extent can convolutional neural networks be used on tiny embedded devices in context of audio classification?". The research is focused on the accuracy of neural networks when deployed on implantable hearing devices. The question is divided into three research subquestions related to described challenges of implantable hearing devices. The first research question addresses the hardware resource constrain challenge and investigates the arithmetic precision reduction of neural networks designed for audio event classification on microcontrollers. The accuracy of an acoustic event dataset is evaluated using two quantization techniques. The second research question is related to the challenge where a model needs to classify both acoustic events and acoustic scenes. A sound event model is evaluated using a sound scene dataset together with accuracy improvement experiments. The last research question investigates the microphone signal challenge where the frequency sensitivity of the input audio is different from the frequency sensitivity of the training audio.

RQ1: ***What is the effect on the accuracy of 8bit post-training quantization and 8bit quantization-aware training on the SB-CNN(-DS) and Stride(-DS) models on the Urbansound8k dataset?***

RQ2: ***What is the effect on the DCASE 2019 dataset classification accuracy when using an audio event classification model SB-CNN(-DS) and Stride(-DS) instead of an audio scene classification model (baseline DCASE 2019)?***

RQ3: ***Which input feature representation among Short-Time Fourier Transform, Mel spectrogram, Constant Q spectrogram achieves the highest Urbansound8k accuracy on the SB-CNN(-DS) and Stride(-DS) models using an implanted MEMS microphone?***

5.2. Research method

This research builds on the master thesis Environmental sound classification on microcontrollers using convolutional neural networks by Nordby [2019]. The work evaluates several optimized CNNs model architectures using the UrbanSound8k dataset on a low-end microcontroller (STM32L476). The models Baseline and Stride-DS-24 achieved the two highest classification scores of respectively 72.3% and 70.9%. The Baseline model is an optimized version of the original SB-CNN model, introduced by Salamon and Bello [2016]. The Stride-DS-24 model is a variant which uses striding instead of max pooling. Both models were evaluated with classic and depthwise separable convolution (-DS suffix) algorithms. These four models will be used as baseline for this research and will be referred to as SB-CNN(-DS) and Stride(-DS). The detailed model architecture is described in section 5.2.2. The python code at https://github.com/jonnor/ESC-CNN-microcontroller was used as starting point. The TensorFlow Lite for Microcontrollers code base currently supports only a subset of the Tensorflow operations which limits the model architecture choice to CNN. In order to answer the research questions, a set of experiment were executed.

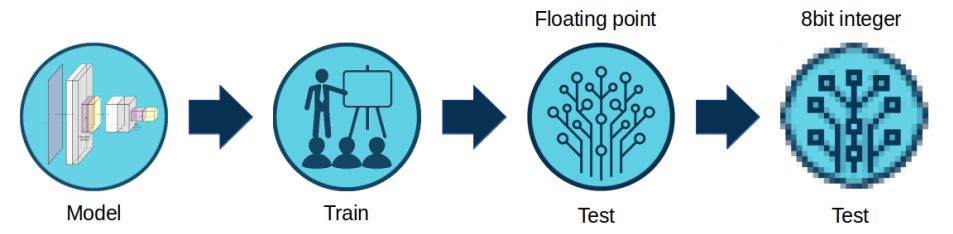

Figure 22: Post quantization experiment flow

RQ1: What is the effect on the accuracy of 8bit post-training quantization and 8bit quantization aware training on the SB-CNN(-DS) and Stride(-DS) models on the Urbansound8k dataset?

Experiments have been executed to compare the overall and detailed accuracy of the quantized models to the floating point models. The floating point model scores are reproduced and serve as a baseline. The process flow followed the first three steps in Figure 22 where the training and testing is executed on a GPU based computer. During the preprocessing, the same data augmentation has been applied as Nordby [2019] using time stretching and pitch shifting for 12 different variations of each UrbanSound8k audio clip. The python framework was extended with post-training quantization and quantization aware training support using Tensorflow 2.3.0 and the Tensorflow Model Optimization Toolkit. Each model training and testing is repeated ten times using a 10-fold validation principle. The overall and detailed accuracy is the averaged result of the ten sessions.

The first part of the experiment post-quantized all baseline floating point models using the 8bit integer conversion for weights and activations. The post quantization process is illustrated in Figure 22. The python framework was extended with two post-training quantization tools; Tensorflow Lite and stm32ai command line tool (CUBE-AI). The default evaluation options were be used for 8bit full integer conversion. The detailed optimization options are out of the scope of this thesis. The representative dataset for scaling of the quantization (section 4.4) is the validation part of the cross validation folds (Refaeilzadeh et al. [2009]). The second part of the experiment retrained all baseline floating point models using quantization aware training. The quantization aware process is illustrated in Figure 23. The python framework was extended with the Tensorflow Model Optimization Toolkit to build network models with quantization aware layers. After the training, the floating point models are quantized using the TensorFlow Lite conversion tool. The 8bit integer models were deployed on an embedded device for evaluation. The X-CUBE-AI software module of the STM32CubeMx development software converted the quantized models into optimized source code. The code is compiled and executed on a STM32L476 microcontroller to provide statistics about the model performance using hardware timers. The tool provides the flash and RAM memory usages as well as the model complexity expressed in multiple and accumulate (MAC) instructions. The model statistics provide validity of actual model quantization.

RQ2: What is the effect on the DCASE 2019 dataset classification accuracy when using an audio event classification model SB-CNN(-DS) and Stride(-DS) instead of an audio scene classification model (baseline DCASE 2019)?

The nature of the sounds that need to be classified on cochlear implant devices overlaps

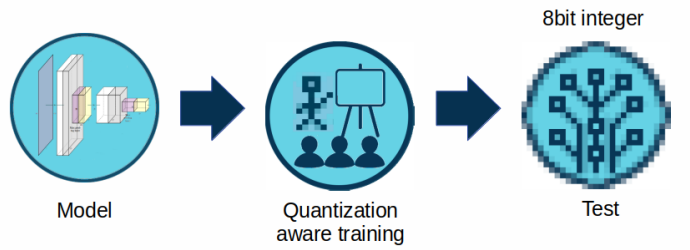

Figure 23: Quantization aware training experiment flow

the acoustic scene classification (ASC) task and acoustic event classification (AEC) task. Neural network model architectures are designed and optimized for a specific task. The thesis baseline models are optimized variants of the SB-CNN model, originally designed for the UrbanSound8k dataset. The model architectures are built for short sound clips ranging from tens of milliseconds to several seconds with classes like dog bark, car horn and others. To evaluate whether the thesis models can be used to classify longer acoustic scenes, the thesis models are trained and evaluted on the DCASE2019 dataset. The DCASE2019 dataset (Mesaros et al. [2019b]) contains 10s sounds clips recorded in public square, park and other scenes. The sounds are recorded with in-ear binaural microphones which is similar to how sound reaches the human auditory system. The dataset is developed for the Detection and Classification of Acoustic Scenes and Events (DCASE) competition organized by the Tampere University of Technology (TUT).

Experiment RQ2-1:
The SB-CNN, SB-CNN-DS, Stride and Stride-DS floating point models are trained using the DCASE2019 dataset. The 10s sound clips are cut into smaller parts to match the 0.7s input width of thesis model. The accuracy results are compared with the baseline model of the DCASE2019 competition which scans the 10s clip in one part.

Can the UrbanSound8k dataset knowledge of the RQ1 floating point models improve the accuracy on the DCASE2019 dataset?
The UrbanSound8k and DCASE2019 dataset are relatively small datasets compared to other machine learning datasets like AudioSet (Gemmeke et al. [2017]). Even when the dataset is synthetically extended using data augmentation, the training epochs are limited to prevent the models to overfit the training data. The idea is to evaluate if both datasets can be used to increase the model accuracy on the DCASE2019 dataset. Instead of initializing the parameters with random values, the parameters are initialized with pretrained values. The experiment will first train the thesis models on the complete UrbanSound dataset using 100 epochs and use it as a start point for the DCASE2019 training. The process flow is illustrated in Figure 24. The transfer of knowledge can only be helpful if the learned features are general. Therefore the experiment is split up to reuse parameters in function of the number of consecutive convolution layers. The fully connected layers are not reused as they are part of the classification for the UrbanSound8k classes.

Experiment RQ2-2:
This experiment will initialize all thesis models with random values and transfer the parameters of all convolution layers the corresponding thesis models trained for 100 epochs

on the complete UrbanSound8k dataset. Next the models are trained for 100 epochs on the DCASE2019 dataset.

Experiment RQ2-3:
This experiment uses a similar setup as RQ2-2 but will transfer only the first and second layer weights and activation values. The third convolution layer and fully connected layer parameters will start from random values.

Experiment RQ2-4:
This experiment transfers the least amount of pretrained parameters. Only the parameters of the first convolution layer are transferred, all other parameters are randomly initialized.

If the model architecture is not optimal for audio scene classification, how can the model be improved?
This experiment investigates the effect on accuracy when changing the model architecture. The input size of the audio window on the thesis models of 0.72s is limited compared to the 10s input size of the DCASE2019 baseline model. The filter size of the thesis models are set to 5x5. Increasing the filter size to 7x7 will capture more time and frequency information within one dot product of a filter. The filters of the consecutive convolution layers will capture more dot products and extend its audio window.

Experiment RQ2-5:
This experiment will increase the 5x5 filter size of the thesis models to 7x7. Due to the striding or pooling layers in the thesis models, the output tensor size of each convolution layer decreases. The output tensor of the second convolution layer of the SB-CNN model is 6x7x48. The filter size of the third convolution layer can not be larger than its input tensor size. Therefore the filter size of the third convolution remains unchanged. The model is initialized with random values.

Experiment RQ2-6:
The model changes of RQ2-5 allow filters to span more audio data. Each input pixel of the flatten layer spans a larger part of the input spectrum. Therefore it is interesting to evaluate if the classification part can improve the classification accuracy by increasing the number of neurons in the fully connected layer. This experiment will use the model of RQ2-5 and increase the number of neurons from 64 to 100.

RQ3: *Which input feature representation among Short-Time Fourier Transform, Mel spectrogram, Constant Q spectrogram achieves the highest Urbansound8k accuracy on the SB-CNN(-DS) and Stride(-DS) models using an implanted MEMS microphone?*

Three different data features are evaluated using the UrbanSound8k dataset. Each feature is trained with the original sounds and with simulated subcutaneous microphone

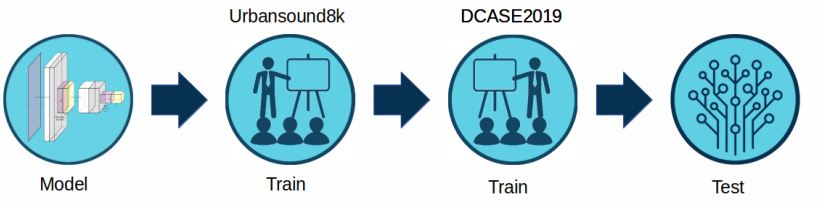

Figure 24: Transfer learning experiment flow

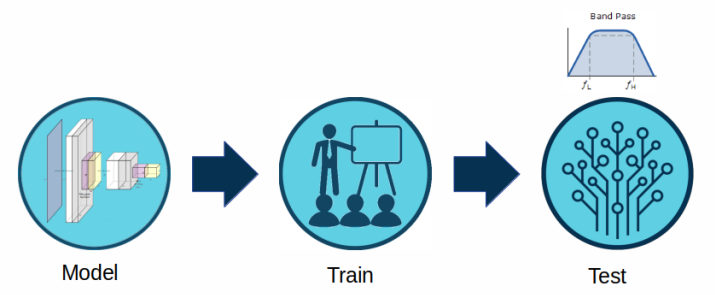

Figure 25: Training experiment flow

sounds. The process flow is illustrated in Figure 25. The UrbanSound8k sound clips are filtered using a 1th order Butterworth band pass filter (BPF) of 100Hz-8kHz and 200Hz-8kHz according to the microphone bandwidth requirements of Calero et al. [2018]. In context of validity a 1th order butterworth high pass filter (HPF) of 100Hz/200Hz and low pass filter (LPF) of 8kHz have been tested to confirm the band pass filtered results.

The experiment was split up into three parts. The first part evaluated the STFT feature. The data feature consists of a 1024 point STFT group which sums the bins into 60 output bins. Next to the unfiltered dataset, the five filters are applied individually to the audio clips to generate a different feature extraction. The models are trained using the unfiltered data and tested on all other filtered data. The second part evaluated the CQT feature. Eight different CQT transformations are generated using different bins per octave configuration (6 to 13). Next to the unfiltered datasets, the five filters are applied individually to the audio clips to generate a different feature extraction. The models are trained using the unfiltered data and tested on all other filtered data. The third part evaluated the Mel feature, where the Mel feature data is reused from RQ1 together with the floating point models. The five filters are applied individually to the audio clips to generate a different feature extraction. The models are tested on the filtered data sets.

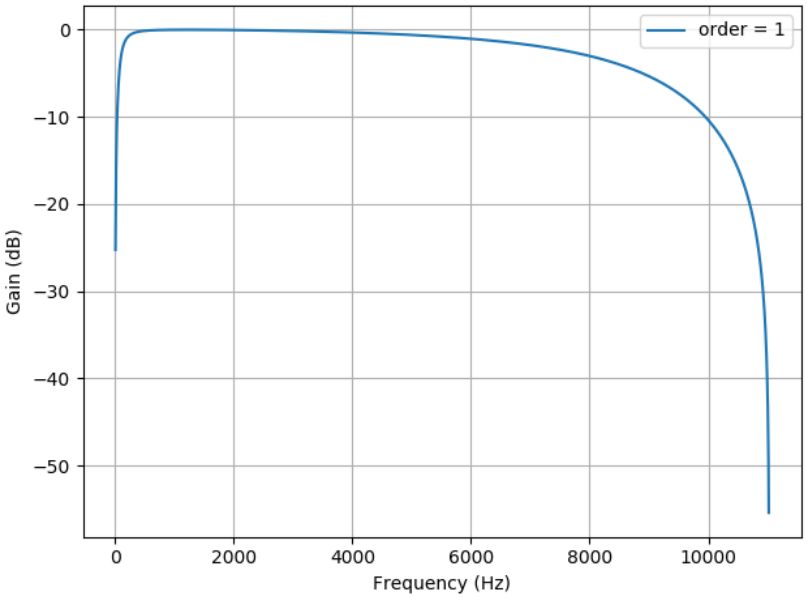

Figure 26: Frequency response of 100Hz-8Khz band pass 1th order butterworth filter

5.2.1. DATASETS

This research trained network models using the UrbanSound8k and DCASE2019 dataset. The taxonomy of the UrbanSound8k dataset is built in context of urban noise pollution in New York City from 2010 to 2013. It contains a collection of short sound events sufficiently detailed to be unambiguous like 'car horn' or 'engine idling'. The dataset is used for the acoustic event classification task. The DCASE2019 dataset consists of sounds that characterize an urban acoustic environment like public square or urban park. The material is used for the acoustic scene classification task.

5.2.1.1 UrbanSound8k dataset

The UrbanSound8k dataset and taxonomy is published by Salamon et al. [2014a]. The dataset is free to use for non-commercial purposes according to the terms of the Creative Commons Attribution Noncommercial License [7]. It contains 8732 sound recordings from freesounds.org with a length between 54ms and 4s. The clips are classified according to 10 classes: air conditioner, car horn, children playing, dog bark, drilling, engine idling, gun shot, jackhammer, siren, and street music. The data is pre-divided into 10 folds for cross-validation. In the context of validation, Salamon et al. [2014a] explicitly ask not to distribute the data randomly but to use the predefined folds for the training as well as results for each fold. Previous publications can therefore be better compared. Table 2 illustrates the data distribution across the different classes. The class car horn, engine idling, gun shot, jackhammer and siren are not evenly distributed among the different folds. The class car horn, gun shot and siren occur 4.91%, 4.28% and 10.64% respectively compared to the other classes that each occur 11.45%, hence the dataset is not fully balanced.

fold	AC	CH	CP	DB	DR	EI	GS	JA	SI	SM
1	100	36	100	100	100	96	35	120	86	100
2	100	42	100	100	100	100	35	120	91	100
3	100	43	100	100	100	107	36	120	119	100
4	100	59	100	100	100	107	38	120	166	100
5	100	98	100	100	100	107	40	120	71	100
6	100	28	100	100	100	107	46	68	74	100
7	100	28	100	100	100	106	51	76	77	100
8	100	30	100	100	100	88	30	78	80	100
9	100	32	100	100	100	89	31	82	82	100
10	100	33	100	100	100	93	32	96	83	100

Table 2: UrbanSound8k sound clip occurance per class

5.2.1.2 DCASE2019 dataset

The Detection and Classification of Acoustic Scenes and Events or DCASE challenge is organized by Tampere University of Technology. For the 2019 challenge, Mesaros et al. [2019a] published the TAU Urban Acoustic Scenes 2019 or DCASE2019 dataset. The dataset contains 14400 sound recordings of 10s recorded in 10 different cities. The clips are classified according to 10 scene classes: airport, indoor shopping mall, metro station, pedestrian

[7] Source https://creativecommons.org/licenses/by-nc/3.0/legalcode

street, public square, street traffic, traveling by tram, traveling by bus, traveling by an underground metro and urban park. For each scene class, recordings were made at 4 to 7 locations in each of the 10 cities. The recordings were made with a Soundman OKM II Klassik / studio A3 with an electret binaural microphone. The microphone consists of two parts and is worn in each ear. The recorded audio is received at the same place where sound waves enter the hearing organ. The dataset is not divided into predefined folds. For this research, the dataset has been divided into ten folds for cross validation. The user is free to choose the best cross validation setup. For this study, the 1440 recordings of the ten cities were randomly distributed over the ten folds for each scene. Each fold has 144 shots for each scene. Table 3 illustrates the distribution of the number of recordings per city in each fold for airport scene.

fold	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Prague	Paris	Stockholm	Vienna	Total
1	11	14	12	10	23	18	13	13	16	14	144
10	14	15	12	14	12	19	14	15	19	10	144
2	16	15	16	15	14	9	16	19	17	7	144
3	11	14	17	19	12	22	12	11	16	10	144
4	9	19	12	16	15	6	21	17	13	16	144
5	16	14	13	17	13	14	13	16	17	11	144
6	12	14	17	10	11	11	20	14	19	16	144
7	17	11	12	16	15	12	13	21	15	12	144
8	12	15	17	12	17	13	8	17	12	21	144
9	10	18	16	16	12	20	14	13	14	11	144

Table 3: Number of city recording per fold for airport scene

5.2.2. BASELINE MODELS

Previous research of Nordby [2019] has investigated the feasibility of deploying convolutional neural network on low-power microcontrollers. Ten models have been optimized and trained using the UrbanSound8k dataset in context of environmental sound classification. The convolutional neural network model with the highest accuracy and the model with the highest performance were selected as baseline for this research, together with their depthwise separable convolution variant. The models are described in more detail in the following sections. More background information on the convolutional networks can be found in section 3.1.1.

5.2.2.1 Salamon-Bello model

The Norby-Baseline model is the optimized version of the original SB-CNN model by Salamon and Bello [2016]. The model will be named SB-CNN in this research. The extracted Mel spectrogram from the sound clips has been reduced from 128 to 60 Mel bands, resulting in an input feature size of 60x31x1. As a consequence of this design choice, the pooling layer has been reduced from 4x2 downsampling to 3x2. Batch normalization has been added after each convolutional layer compared to the original model.

Figure 27 depicts the model architecture of the optimized SB-CNN model. The low-level feature extraction is created by convolution layer 1 with classic convolution layer using same padding with 24 filters of filter size 5x5. The feature maps (white) are batch normalized (grey), downsampled using a 3x2 max pooling layer (purple) and activated by a rectified linear unit (yellow). The tensor output at this stage is 20x15x24.

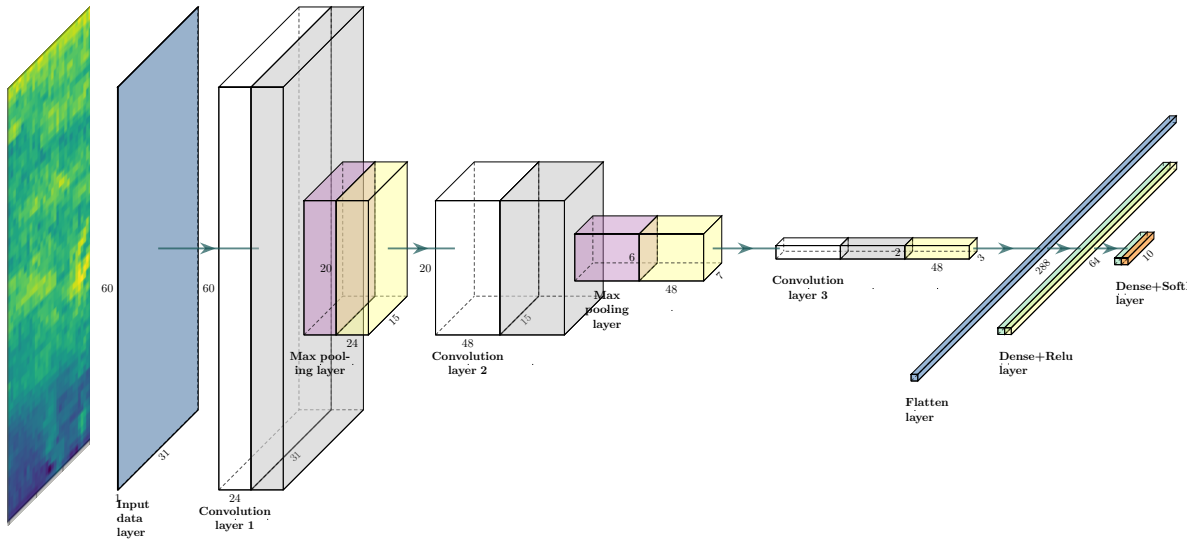

Figure 27: Optimized Salamon-Bello convolutional neural network model.

The higher-level feature extraction is performed by convolution layer 2 with classic convolution layer using same padding with 48 filters of filter size 5x5. The feature maps (white) are batch normalized (grey), downsampled using a 3x2 max pooling layer (purple) and activated by a rectified linear unit (yellow). The tensor output at this stage is 6x7x48.

The high-level feature extraction is performed by convolution layer 3 with classic convolution layer using valid padding with 48 filters of filter size 5x5. The feature maps (white) are batch normalized (grey). Due to the padding, the tensor output at this stage is 2x3x48.

Next, the flatten layer converts the 3D feature data of 2x3x48 into a 1D input vector of 288 parameters in order to feed the data into the classifier. All the input parameters are connected to the fully connected or dense layer (section 3.2.2) of 64 neurons which consists of 18496 parameters. Each of 64 neurons is connected to the 10 neurons output layer which are activated using a softmax function (section 3.2.3) to output to probabilities for the 10 classes of the UrbanSound8k dataset.

The SB-CNN model uses classic convolution in its convolution layers. A SB-CNN-DS variant is created using the same model architecture but using depthwise separable convolution in the second and third convolultion layer. The computational complexity is significantly lower due to the reduced number of multiplications and learnable parameters (section 3.1.2).

5.2.2.2 Stride model

The Stride model is based on the previously discussed SB-CNN model. The main difference is the absence of the max pooling layers. The downsampling is replaced by a striding step in the convolution layers. The Keras framework does not support the same 3x2 downsampling as in the SB-CNN model. The stride height and width must be uniform, the max pooling of 3x2 is replaced by 2x2 downsampling. The number of filters grows 50% per convolution layer.

Figure 28 depicts the model architecture of the strided SB-CNN model named Stride. The low level feature extraction is created by convolution layer 1 with classic convolution layer (section 3.1.1) using same padding with 22 filters of filter size 5x5. The downsampling action is replaced by a stride step of 2x2. The feature maps (white) are batch normalized (grey) and activated by a rectified linear unit (ReLU). The tensor output at this stage is 30x16x22.

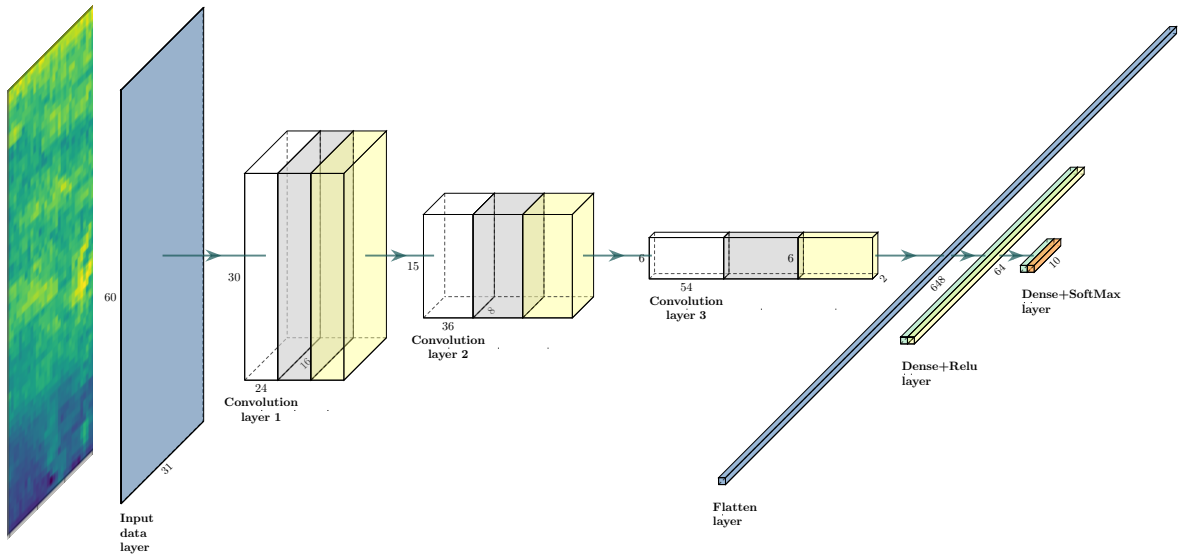

Figure 28: Strided-DS Salamon-Bello convolutional neural network model.

The higher-level feature extraction is performed by convolution layer 2 with classic convolution layer using same padding with 36 filters of filter size 5x5. The feature maps (white) are batch normalized (grey) and activated by a rectified linear unit (ReLU). The downsampling is replaced by a stride step of 2x2. The tensor output at this stage is 15x8x33.

The high-level feature extraction is performed by convolution layer 3 with classic convolution layer using valid padding with 54 filters of filter size 5x5. The feature maps (white) are batch normalized (grey) and activated by a rectified linear unit (ReLU). The downsampling is replaced by a stride step of 2x2. The tensor output at this stage is 6x2x49.

Next, the flatten layer converts the 3D feature data of 6x2x49 into a 1D input vector of 588 parameters in order to feed the data into the classifier. All the input parameters are connected to the fully connected or dense layer (section 3.2.2) of 64 neurons which consists of 37696 parameters. Each of 64 neurons are connected to the ten neurons output layer which are activated using a softmax activation function (section 3.2.3) to output probabilities for the ten classes of the UrbanSound8k dataset.

The Stride model uses classic convolution in its convolution layers. A Stride-DS variant

is created using the same model architecture but using depthwise separable convolution in the second and third convolution layer. The computational complexity is significantly lower due to the reducted number of multiplications and learnable parameters (section 3.1.2). The number of filters in the first layer starts from 24 filter instead of 22 filters in the non-strided variant and grows 50% per convolution layer.

5.3. TRAINING

The training settings used during this research are identical to the research baseline of Nordby [2019]. The training is performed with a learning rate of 0.005 on mini-batches of 400 audio recordings. The optimizer function is a Stochastic Gradient Decent (SGD) with 0.9 Nesterov momentum. A random audio window is chosen for each recording to apply the time-shifting data augmentation. Each training run consists of 100 epochs. Each epoch processes 30000 training samples and 5000 validation samples. Because the training data is split up in 10 folds using cross validation, a model training session consists of 10 training runs. The best model for each fold is selected and combined into an average accuracy score per session. The model accuracy of each experiment is the average result of 10 training sessions unless noted differently.

The training hardware is built on an Intel i7-9700K CPU core using 32GB 3200-16 Vengeance LPX RAM memory. For the training acceleration, an MSI 8GB D6 RTX 2070 Super GPU was chosen. An SSD 970 EVO PCIe hard disk drive was directly interfaced by the Asus ROG STRIX Z390-E Gaming motherboard. On this machine, a ten fold training session from RQ1 takes approximately 3.5h for 1 model with five fold training jobs running in parallel.

5.4. EVALUATION

The evaluation of the network accuracy is performed with the test set of each cross-validation fold. The test set is a predefined subset of the audio files from the UrbanSound8k or DCASE2019 dataset. An audio file can be up to 4s long and is adapted to the input of the network. The input tensor has a size of 60x31x1 which corresponds to a spectrogram of 60 bands over 31 frames. Each sound clip is split into windows of 31 frames which corresponds to 730ms audio. When the audio information is shorter than 730ms, padding is applied to fill the data with zeros.

Figure 29 illustrates the dog bark sound file 100795-3-0-0.wav that is split into 6 windows. The class prediction score of each window is collected and for each audio file the average of all window prediction scores is made. The highest scoring class is the final prediction. This technique is called mean voting. The prediction outcome of all files is presented in a confusion matrix. The matrix indicates how many times the correct prediction is made per class and which class was voted when the prediction was wrong. Figure 30 depicts the confusion matrices where the relationship between the real presented audio labels on the Y-axis and the predicted labels on the X-axis. Each horizontal line therefore contains the score distribution of all audio files of a certain class. In Figure 31, the results are normalized showing the scores relative to 100 percent. The diagonal line represents the intersection between the real result and the predicted result. The total accuracy score of a model is the ratio between the sum of the numbers on the diagonal line (the correct predictions) and all numbers.

software

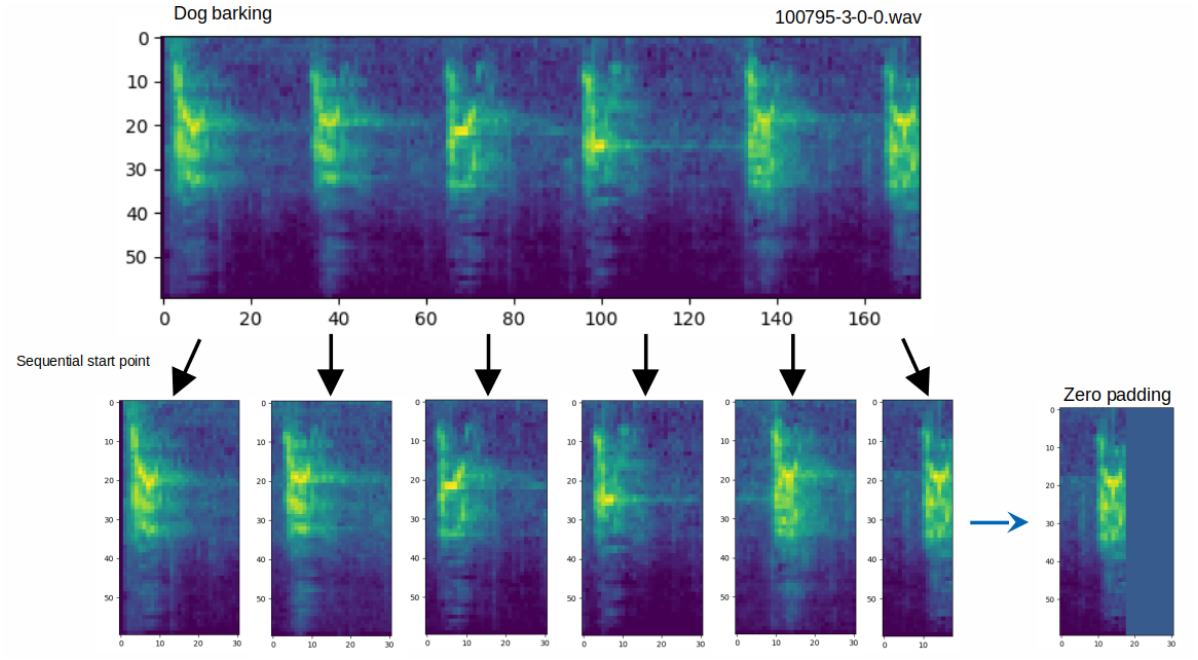

Figure 29: Dog bark sound clip of 4s split into classification windows of 720ms.

5.5. SOFTWARE TOOLBOX

The linux software setup for a machine learning development environment is complex. The research domain is very active, where the speed of software tool developments and the pace of releases are high. There are many dependencies between the mutual software components, hence a detailed description is important to get a working setup. This section provides an overview of the development environment.

The framework to train, evaluate and test the models is based on the research of Nordby [2019]. The code is hosted at Git. The Python framework is used to reproduce the results of baseline network models. Subsequently, the framework was expanded in the context of quantization and support for other datasets and features. All floating point models are

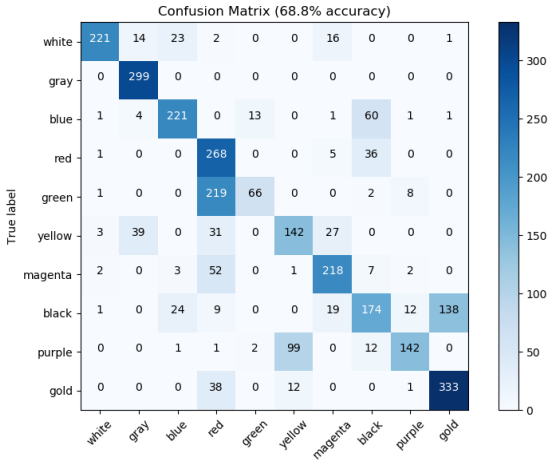

Figure 30: Confusion matrix

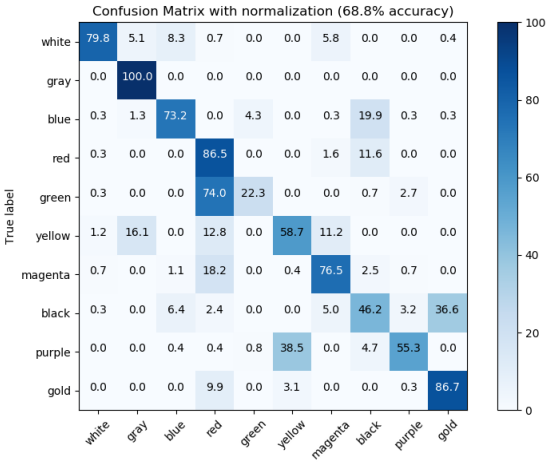

Figure 31: Normalized confusion matrix

created in Keras (v2.4.0), the quantization aware variants are created with the Tensorflow Model Optimization toolkit 0.5.0. All models are trained using Tensorflow v2.3.0. The Tensorflow Lite module is used to convert the floating point models to integer models via the OptimizeForSize option. The representative dataset for the post quantization is the validation set of each data fold.

The STM32 Cube software (v5.6.1) is used to evaluate the models at the microcontroller level. The X-CUBE-AI (v5.2.0) extension is a set of libraries and plugins to deploy neural networks on STM32 microcontrollers. The software cannot train models itself but supports trained models from Tensorflow, Keras, PyTorch and other machine learning frameworks. The models are integrated in low-level C code and compiled for an ARM Cortex architecture. The software GUI has functionalities such as performance measurement and validation. The performance measurement function deploys the network on the microcontroller and uses internal hardware timers to measure the inference time. The analysis option indicates how much RAM / flash memory is required or what the required computation power is per network layer. The STM32 software also offers the possibility to perform post quantization. However, the option is marked as deprecated in the latest version. Post quantization for the STM32 model was performed with the stm32ai Neural Network Tool v1.4.0 command line tool. The calibration data set used on the validation part of each data fold. The tool does not generate a quantized output model but generates a JSON file with the scale and offset information that, together with the original floating keras model, is used by the STM32 software to generate target code directly.

The validation functionality offers the possibility to test the C code model on a PC (x86 architecture) so that the speed is not slowed down by the transfer time between the PC and the microcontroller. A numpy (v1.17) array with data and labels can be used as input for the software to measure accuracy. The x86 model is not directly accessible in the current software version. There is no support for mean voting, which means that all sound windows of a clip must be fed individually into the software using a separate numpy file. For each run the software recompiles the network model C code, making it not feasible in time to determine the accuracy in the same way as the baseline. That is why it was decided not to spend any effort on the accuracy of the post quantization of the STM32 model in context of mean voting accuracy.

The Python version used is v3.6.9. All python package dependencies are captured in an environment configuration file and managed by python package management tool Miniconda v4.8.2. All tools are installed in docker (v19.03.8) images. The GPU GTX2070 acceleration is activated by the CUDA v10.2 in combination with the nVidea 440.59 driver. Pycharm v2020.2 professional is used as a Python integrated development environment to debug python code running on remote docker images. The continuous integration framework Teamcity v2020.1.3.78866 has been used to deploy the command line tasks for training, testing and reporting. The teamcity server deploys the development work from a version control server (Github) to teamcity test agents to execute preconfigured command lines within the docker image. It has been set up to systematically capture all artifacts generated by training runs and deploy artifacts as a dependency for a test or report processes.

6. RESULTS

All objective evidence generated during the experiments are documented in this section. Each section relates to a research question. Section 6.1 documents the result of experiments in order to answer RQ1 regarding to the effect on the accuracy of 8bit post-training quantization and 8bit quantization-aware training on the SB-CNN, SB-CNN-DS, Stride, and Stride-DS-24 models using the Urbansound8k dataset. Section 6.2 documents the experiments regarding to RQ2 where the floating point models designed for audio event classification are used in context of audio scene classification using the DCASE2019 dataset. The section 6.3 handles the experiments regarding to RQ3 where STFT, CQT and Mel feature are compared using the Urbansound8k dataset.

6.1. RESULTS RQ1

The first challenge in the research design is related to hardware resource constraints. Implanted medical devices are tiny devices where the battery life constrains the available memory, processing power and processor complexity. The challenge can be addressed by model quantization which reduces the arithmetic precision from 32bit floating point to 8bit integer calculations. The benefit is that integer calculations are faster. The parameters take less memory size and the memory bandwidth is reduced. The drawback is the rounding error which is introduced by quantization. RQ1 investigates two quantization approaches on small neural networks: '***What is the effect on the accuracy of 8bit post-training quantization and 8bit quantization-aware training on the SB-CNN(-DS) and Stride(-DS) models on the Urbansound8k dataset?***.

The experiments evaluate the SB-CNN and Stride floating point models using classic convolution and the optimized depthwise separable convolution variant. The models are trained using TensorFlow (TF) framework on the UrbanSound8k dataset. The first experiment performs post quantization (PQ) of the trained floating point models using the TensorFlow (TF) model optimization toolkit and the stm32ai command line tool. The representative dataset used to calibrate the quantization is the validation data fold. The second experiment retrains the models using training aware quantization. The accuracy is evaluated using TF on computer level and reported using overall comparison with a deeper analysis of the confusion within the predictions. All models are compiled using the X-CUBE-AI libraries and deployed on the STM32L476 microcontroller to measure the inference performance and memory consumption.

6.1.1. OVERALL ACCURACY

Table 4 provides an overview of the accuracy variation for the floating point baseline models.

Model Test Accuracy				
	SBCNN	SBCNN-DS	Stride	Stride-DS
Session 1	71.0%	71.0%	69.8%	67.8%
Session 2	72.5%	68.9%	71.3%	68.5%
Session 3	71.1%	69.1%	70.0%	68.6%
Session 4	69.6%	70.7%	70.7%	68.8%
Session 5	72.9%	69.0%	69.6%	68.8%
Session 6	71.5%	70.1%	70.3%	69.5%
Session 7	70.3%	70.0%	69.8%	70.9%
Session 8	72.6%	70.6%	68.9%	69.0%
Session 9	71.7%	71.1%	70.7%	68.4%
Session 10	70.8%	70.8%	70.1%	69.5%
Minimum	69.6%	68.9%	68.9%	67.8%
Average	**71.4%**	**70.1%**	**70.1%**	**69.0%**
Maximum	72.9%	71.1%	71.3%	70.9%
Baseline	72.3%	70.2%	68.3%	70.9%

Table 4: Floating point model test accuracies

The SB-CNN model achieved the highest test accuracy score of 71.4%. The depthwise separable convolution variant SB-CNN-DS scored 1.3% less and achieves 70.1% test accuracy. The stride model achieves the same accuracy 70.1% while its depthwise separable convolution variant Stride-DS scored 1.1% less and arrived at 69.0% test accuracy. The positive tolerance ranges between 1 to 1.9%, while the negative tolerance ranges from 1.2 to 1.5%.

Model	Floating point	TF-PQ	TF-QAT
SBCNN	71.4%	71.3%	71.2%
SBCNN-DS	70.1%	70.0%	69.9%
Stride	70.1%	70.1%	70.1%
Stride-DS	69.0%	68.9%	68.7%

Table 5: Test accuracy results using voted prediction.

Table 5 reports the average accuracy results for the floating point and integer models using mean voting strategy (section 5.4). The test accuracy of the TFLite post quantizated (TF-PQ) models of SB-CNN, SB-CNN-DS and Stride-DS score 0.1% lower than their keras floating point models. The Stride TFLite post quantizated model actually scores the same 70.1% test accuracy compared to the floating point variant. The quantization aware models of SB-CNN and SB-CNN-DS indicate a 0.2% drop in accuracy versus the keras floating point model. The TFLite quantization aware (TF-QAT) Stride model scores the same accuracy of 70.1% while the depthwise separable convolution variant scored 0.3% lower.

The STM32 post quantized model is not reported because it can not be evaluated using voted prediction strategy out of the box. The STM32 tool creates a post quantized model

by exporting a floating point keras model and a JSON file containing the scale and offset information for the model. The STM32 validation tool can only take in a set of test data windows with a corresponding class for each window. It does not allow to extract a list of predictions and apply voting strategy. Therefore the validation score cannot be compared with the baseline in the overall accuracy results and confusion matrixes. However, for comparison, all models were tested using the STM32 tool without mean voting. The STM32 post quantization test accuracy can be compared relative to the other models.

The keras models score 12.1% to 12.5% less due to the voting strategy. The TFLite post quantization model scores are 0.1 to 0.2% lower than the keras models. The TFLite quantization aware model scores are 0.1 to 0.2% lower than the TFLite post quantization models, except for the Stride model. The STM32 post quantization models are 0.1% to 0.3% lower than the keras models.

Model	Floating point	TF-PQ	TF-QAT	STM32-PQ
SBCNN	59.3%	59.2%	59.0%	59.0%
SBCNN-DS	58.4%	58.2%	58.1%	58.2%
Stride	57.1%	57.1%	57.3%	57.0%
Stride-DS	56.5%	56.4%	56.3%	56.3%

Table 6: Test accuracy results using non-voted prediction.

6.1.2. DETAILED ACCURACY

Table 7 lists the detailed accuracy for all models. The classes with the highest accuracy are gun shot (91.7-94.1%), car horn (83.9-85.9%) and dog barking (82.1-85.8%). The least scoring class is air conditioning (41.6-47.3%) which is misclassified 18.7-21.2% as engine idling. Both drilling and engine idling class are respectively 14.9-15.7% and 17.0-20.3% wrongly classified as jackhammer. The depthwise convolution variants score 3.9-4.8% less on the children playing class compared to the classic convolution. The street class achieves 76.7& accuracy on the SBCNN model but drops 4.5 to 6.9% on all models.

Class	SBCNN	SBCNN-DS	Stride	Stride-DS
Air conditioner	41.6%	47.3%	43.8%	44.7%
Car horn	85.9%	83.9%	86.3%	84.3%
Children playing	80.0%	75.2%	78.4%	74.5%
Dog bark	85.8%	83.2%	83.8%	82.1%
Drilling	64.4%	62.4%	61.6%	60.5%
Engine idling	59.6%	56.6%	62.0%	60.4%
Gun shot	94.1%	92.6%	94.4%	91.7%
Jack hammer	67.7%	70.5%	62.6%	64.1%
Siren	81.3%	79.5%	81.0%	81.0%
Street music	76.7%	72.2%	71.8%	69.8%
Overall	71.4%	70.1%	70.1%	69.0%

Table 7: Detailed accuracy floating point results using voted prediction.

Figure 32 provides detailed accuracy in which the post quantization and quantization aware variants are compared with the floating point model. The post quantization matrix only includes TFLite variants and not the STM32 variant as discussed in section 5.5. Each matrix is the average result of 10 train sessions. The matrix of the floating point model is depicted with blue background. The matrices of the post quantization and quantization aware model are relative compared to the floating point accuracy. The green and purple colors indicate where the model scores better or worse. The scale corresponds to the maximum deviation found in the quantized models.
The TF-PQ models lose 0.1% or less using 8bit integer quantization compared to the floating point models. The confusion matrix indicates minimal differences over the 10 classes. The highest accuracy loss for all TF-PQ models is found on the classification of gun shot which achieved the highest predicted score in the floating point models. The accuracy drop for the models SB-CNN, SB-CNN-DS, Stride and Stride-DS is respectively 0.8, 1.2, 0.3 and 1.2%. The TF-QAT models score 0.3% or less on overall accuracy, but there are more individual differences compared to floating point models. The differences on the diagonal axis indicate that the model scores better on some classes and worse on others.

Figure 33 provides a comparison of the confusion matrix of the quantization aware models of SB-CNN, SB-CNN-DS, Stride and Stride-DS. The summation of all the absolute deviations are plotted in one confusion matrix with the same true and predicted labels. It is not a standardized accuracy measurement but provides an indication of where the largest differences are found. Two groups of deviations are visible. The first group of deviations are found along the diagonal axis which represents the correct inference where true and predicted label match. The second group of deviations is found in the predictions for air conditioner, the overall lowest scoring class.

6.1.3. Memory size

This section reports the required parameter memory for flash and RAM storage in context of quantization. The weights used inside the model are read-only and can be stored in flash memory. The results of the activations and intermediate results are stored in RAM memory. On flash perspective, the SB-CNN model requires 415kB of flash size to store the parameters. The depthwise separable variant SB-CNN-DS requires 98.4kB which is 4.2 times less parameters. The Stride model requires 381.2kB of flash size to store the parameters. The depthwise separable variant Stride-DS requires 184.6kB which is 2.1 times less parameters. The post quantized models require 8bit storage for integers instead 32bit storage for floating point numbers. Hence the quantized models SB-CNN, SB-CNN-DS, Stride and Stride-DS require 3.97, 3.87, 3.97 and 3.97 times less flash memory respectively. By applying the depthwise separable and convolution, the SB-CNN-DS model is 16.35 times smaller while the Stride-DS model 8.13 times smaller than the keras floating point variants.

On RAM perspective, the SB-CNN model requires 48.6kB of volatile memory in the floating point keras model. The TFLite post quantization and quantization aware models require 25.5kB RAM which is 1.9 times less memory. The STM32 post quantization model requires even less RAM memory and ends up at 19.7kB which is 2.48 times less RAM memory than the floating point variant. The depthwise separable variant SB-CNN-DS requires 2.41 times less RAM memory for the TFLite variants than its keras floating point variant. The STM32 SB-CNN-DS model requires 13.3kB RAM which is 3.6 times less memory than the original floating point model. The Stride model has 49.7kB of volatile memory in the floating point keras model. The TFLite post quantization and quantization aware models

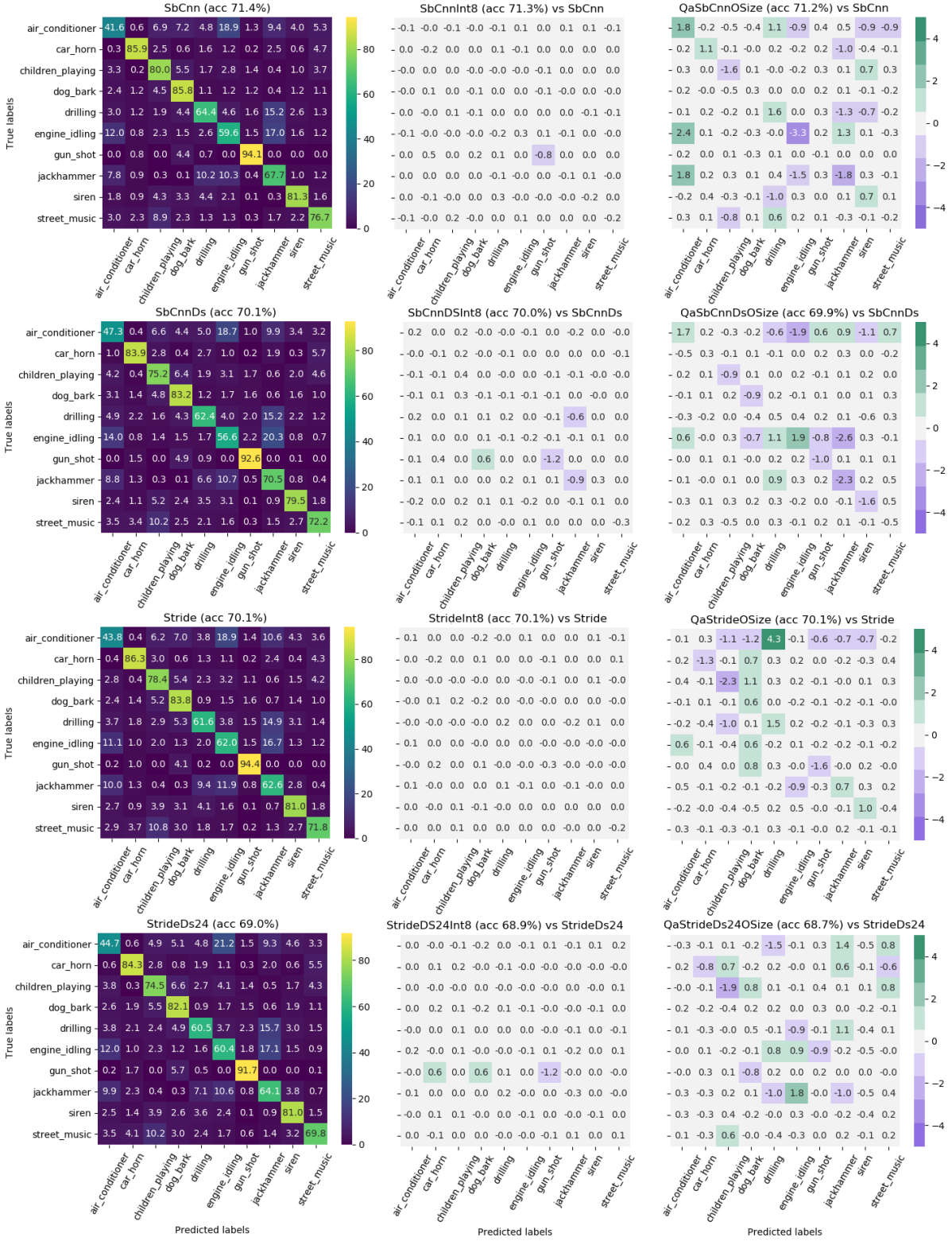

Figure 32: Confusion matrix results of floating point models versus quantized 8bit integer models.
Top to bottom; SB-CNN, SB-CNN-DS, Stride, Stride-DS
Left to right; keras, post quantization, quantization aware model

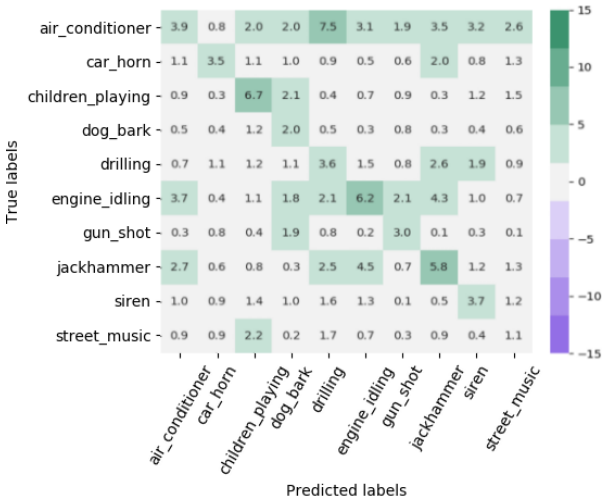

Figure 33: Quantization aware model confusion differences versus floating point models

require 25.5kB RAM which is 1.95 times less memory. The STM32 post quantization model requires even less RAM memory and ends up at 19.7kB which is 2.51 times less RAM memory than the floating point variant. The depthwise separable variant SB-CNN-DS requires 2.55 times less RAM memory for the TFLite variants than its keras floating point variant. The STM32 SB-CNN-DS model requires 15kB RAM which is 3.52 times less memory than the original floating point model.

The overall model size illustrates the total amount of memory required, combining flash and RAM memory. The largest floating point model is the SB-CNN with a model size of 463.75kB while the TFLite integer variant of 129.91kB is 3.57 times smaller. The SB-CNN-DS model has a total memory size of 146.5kB for the floating point model with a 3.24 smaller TFLite integer model of 45.31kB. The second largest model is the Stride model with 430.8kB with a 3.55 times smaller TFLite integer variant of 121.33kB. The Stride-DS floating point model size is 237.78 with a TFlite integer variant size of 67.65kB which is 3.51 times smaller. All integer models are below 128kB (red bar), except for the TFLite SB-CNN variants which are just above the limit with a model size of 129.91kB.

Model	Variant	Acc	MACCs	Inference time	ROM	RAM	Total size
SBCNN	Keras	71.4%	10.2M	1114ms	415k	48k6	464k6
	TF-PQ	71.3%	10.2M	425ms	104k	25k6	129k6
	TF-QAT	71.2%	10.2M	425ms	104k	25k6	129k6
	STM32-PQ	-	10.2M	441ms	104k	19k7	123k7
SBCNN-DS	Keras	70.1%	1.75M	336ms	98k4	48k1	146k5
	TF-PQ	70.0%	1.74M	127ms	25k4	19k9	45k3
	TF-QAT	69.9%	1.76M	127ms	25k4	19k9	45k3
	STM32-PQ	-	1.74M	119ms	25k4	13k3	38k7
Stride	Keras	70.1%	2.98M	335ms	381k	49k7	430k7
	TF-PQ	70.1%	2.97M	182ms	95k8	25k5	121k3
	TF-QAT	70.1%	2.97M	182ms	95k8	25k5	121k3
	STM32-PQ	-	2.97M	165ms	95k8	19k7	115k5
Stride-DS	Keras	69.0%	557k	105ms	184k	53k1	237k1
	TF-PQ	68.9%	554k	40.1ms	46k9	20k8	67k7
	TF-QAT	68.7%	554k	40.1ms	46k9	20k8	67k7
	STM32-PQ	-	540k	37.4ms	46k9	15k1	62k

Table 8: Quantisation results on UrbanSound8k dataset

6.1.4. INFERENCE PERFORMANCE

Table 8 reports two parameters in terms of performance; computational complexity (MACC) and inference time. On MACC perspective requires the SB-CNN model 10185k MACCs to run one inference in a keras model. The depthwise separable convolution variant requires 1746k MACC operations which is 5.8 times fewer calculations. The Stride model requires 2980k MACCs to run one inference in a keras model. The depthwise separable convolution variant requires 556k MACC operations which is 5.4 times fewer calculations. The post quantization or quantization aware models have a comparable number of operations compared to their floating point model variant. The cost of the computional complexity saving has an impact on the accuracy. The SB-CNN model drops 1.3% test accuracy when depthwise separable convolution is used while the Stride model drops 1.1% in accuracy. The number of parameters in the model does not change hence the count of convolution and dense layer operations are the same. However, the storage size of the parameters is different in the quantizated model variants.

On inference time perspective, requires the SB-CNN model more than 1 second to perform one inference while the depthwise separable variant runs 3.32 times faster. The integer variants of the SB-CNN floating point model 2.62 times faster and executes in 336ms. The integer variants of the SB-CNN-DS floating point model 2.64 times faster. The Stride model performs one inference in 335ms while the depthwise separable variant runs 3.19 times faster. The integer variants of the Stride floating point model 1.84 times faster. The integer variants of the Stride-DS floating point model 2.63 times faster. The inference time results are proportional to the number of MACC operations of the models.

6.1.5. DISCUSSION

The test accuracy scores for the baseline floating point models SB-CNN, SB-CNN-DS, Stride and Stride-DS are comparable with the research results published by Nordby [2019]. In

context of reliability, the training sessions were repeated 10 times. The test accuracy is slightly different for each train session as differs per test session. This is because the network starts with random values and because time shifting augmentation is applied to the dataset during training. The average accuracy results converge to a point where the depthwise separable convolution variants score 1.3% lower for SB-CNN and 1.1% lower for the Stride model. This is a consequence of the number of MACCs being 5.8 and 5.4 times lower respectively, compared to the classic convolution models. In the baseline research of Nordby [2019], the depthwise separable variants score 2.1% lower and 2.6% higher respectively. Although these scores were reproduced in this research, the overall 10 session mean provides a more precise result on the actual accuracy. This underlines the importance of repeating the experiments.

The TFLite post quantization technique resulted in a maximum 0.1% accuracy loss. The confusion matrix for each model after post quantization shows minimal differences from the floating point variant. The number of class accuracy deviations above 0.5% for SBCNN, SB-CNN-DS, Stride-DS are 1, 4 and 3 respectively. The Stride model had no deviation above 0.5%. It is worth noting that the greatest deviation after post quantization occurs at the place where the floating point model scores highest. The gun shot class achieves the highest score with more than 90% for all models. Exactly at the real-predicted intersection of the gun shot class, each post quantization model has the greatest loss. The accuracy loss for correct predictions from the gun shot class for the models SB-CNN, SB-CNN-DS, Stride and Stride-DS is 0.8, 1.2, 0.3 and 1.2%, respectively. Next to the gun shot class, only the SB-CNN model has an accuracy loss for the jackhammer of 0.9%. The other losses are negligible.

The quantization aware technique has 0.1% overall accuracy loss more than post quantization on the SBCNN models. For Stride model there is no loss for both techniques. For the Stride-DS model, there is 0.2% more loss than post quantization. It is noteworthy that the confusion matrix indicates much more accuracy differences on the diagnosis where the correct predictions are located. Figure 33 gives an absolute summation of all deviations of the quantization aware models from the floating point models.

The theoretical computational complexity does not change by applying post quantization or quantization aware training. The number of MACCS operations remains the same for each model variant. The number of weights also remains the same but the size of each weight is smaller. The 8bit integers take up 4 times less space than the floating point numbers. Therefore, the flash memory usage for each model drops by a factor of 4. The RAM for activations also shrinks 4 times but is less affected due to the internal scratchpad working memory that the interference engine needs to store intermediate results. The RAM for the STM32 post quantization models is smaller than the required TFLite memory because the input tensor is in integer format instead of floating point. The disadvantage is that the scaling parameters must be known outside the model logic, but it can be more efficient if the preprocessing is already running in integer format. Depending on the type of network, the overall memory size of the 8b integer model is 3.23 to 3.57 times smaller for the TFlite models. All quantized models are 6.67kB to 89.4kB below the 128kB memory limit except for the SB-CNN TFLite model which is 1.19kB larger. The models are eligible for deployment

on tiny embedded devices and meet the research model size goal. The smallest model is the SB-CNN-DS model with a total memory size of 38.6kB when the inputs are 8bit integer or 45.3kB with floating point inputs. The headroom provides an opportunity to grow the model accuracy by for example increase the number of filters.

The fastest model is the quantized Stride-DS model with 37.4ms for a model with integer inputs and 40ms with floating point inputs. The inference time is different between the floating point and integer models while the number of MACC operations remains the same. This is because integer operations are much faster and more efficient. Despite the fact that the STM32476 microcontroller used a dedicated floating point hardware unit during the measurements to accelerate these operations, the integer models remain faster. The SB-CNN and Stride integer models run 3.19 to 3.32 times faster than the floating point variant. The depthwise separable integer models run 2.63 times faster. An integer Stride-DS model can do 10.6 times more inferences than an integer SBCNN model. The accuracy loss of the Stride-DS model could be improved by analyzing more audio in the same amount of time. The downside is that the audio preprocessing needs to be faster hence will consume more power.

6.1.6. Conclusion

The network models SB-CNN, SB-CNN-DS, Stride and Stride-DS have been quantized from 32bit floating point precision to 8bit integer precision and evaluated using the UrbanSound8k dataset. The effect on classification accuracy of 8bit post quantization and 8bit quantization aware training is negligibly small compared to the floating point models. TP-PQ models score equal or slightly better than TP-QAT models. It has been suggested by Jacob et al. [2017] that the post-quantization approach works well on large network models with considerable representational capacity but would lead to a significant accuracy drop on smaller models. The model sizes of referenced large networks are 98MB for ResNet-50 (He et al. [2016]), 92MB for Inceptionv3 (Szegedy et al. [2015]) versus the MobileNet 16.8MB Howard et al. [2017b] and MobileNet SSD 23MB. However, this is not the case for the baseline models of this thesis with models ranging from 0.15MB to 0.44MB where post quantization performs better of equal. There is no significant effect on accuracy comparing the classic convolution models to the depthwise separable convolution regardless of having 5.3 to 5.8 fewer MACC operations. The accuracy loss for QAT models is comparable with the results reported by Krishnamoorthi [2018].

The experiments have demonstrated a substantial 3.24 to 3.57 model size reduction, bringing three models far below the 128kB memory limit of resource scarce devices while keeping the accuracy loss within 0 to 0.2%. Because the models perform the same inference 1.84 to 3.32 times faster, the battery powered devices can go back to sleep earlier which is beneficial for power consumption. Post quantization performed overall better or equal than training aware quantization and does not require model retraining. Quantization is a feasible optimization technique to bring highly optimized networks to tiny embedded devices like wearables or implanted medical devices in context of audio classification.

In context of model deployment, the three important criteria are accuracy, power consumption and memory size. For applications where accuracy is preferenced, the SB-CNN model is the best choice. The model achieves an accuracy of 71.3% for a model size of 129.88kB. The most energy-efficient model is the Stride-DS model with the smallest inference time of 40.0ms. The model also achieves the lowest accuracy of 68.9% among all mod-

els. Because the model is fast and output ten times more inferences in the same amount of time than the model with the most accuracy. The Stride-DS model can therefore potentially achieve higher accuracy by processing greater audio overlap. The extra consumption for the audio preprocessing must then be taken into account. When memory size is the most important constraint, the SB-CNN-DS model takes up the least space with 45.31kB with an accuracy of 70.0%. Due to the combination of speed and memory size gain, it is possible to deploy the models on a smaller hardware class. Because the integer model no longer requires a floating point unit, the used CortexM4 microcontroller architecture can be exchanged for a CortexM3 architecture, which is advantageous for the cost price.

6.2. RESULTS RQ2

The second question is related to the nature of the sounds the model needs to classify in context of implantable hearing devices. The classification overlaps the acoustic event classification and acoustic scene classification tasks. The thesis models are originally designed for the UrbanSound8k dataset which contains short individual sound events. Acoustic scenes are longer audio recordings which are built up from multiple sound events that characterize the environment. The DCASE2019 dataset contains audio scenes like public square, park or others. RQ2 investigates scene classification accuracy of the thesis models using the DCASE2019 dataset. Two subquestions explore potential accuracy improvements.

What is the effect on the DCASE 2019 dataset classification accuracy when using an audio event classification model SB-CNN(-DS) and Stride(-DS) instead of an audio scene classification model (baseline DCASE 2019)?
Experiment RQ2-1 evaluates the floating point thesis models on the DCASE2019 dataset and compares the results with the DCASE2019 baseline competition model.

Can the UrbanSound8k dataset knowledge of the RQ1 floating point models improve the accuracy on the DCASE2019 dataset? Three experiments are executed to combine both datasets for the training of the thesis models. First, the models are trained on the complete UrbanSound8k dataset. Next, the learned features are transferred into a new random initialized model which is used as starting point for the training with DCASE2019 dataset. Experiment RQ2-2 reuses the weights of 3 convolution layers and only retrains the dense layer. Experiment RQ2-3 reuses the weights of 2 convolution layers and retrains the last convolution layer and the dense layer. Experiment RQ2-4 reuses the weights of 1 convolution layer and retrains the last two convolution layer and the dense layer.

If the model architecture is not optimal for audio scene classification, how can the model be improved? The thesis model architectures are improved in two experiments. The first experiment RQ2-5 increases the filter size from 5x5 to 7x7 for the first two convolution layers. This increases the input data span for each deeper layered pixel. The second experiment RQ2-6 improves the dense layer classification capability by increasing the number of neurons from 64 neurons to 100 neurons together with 7x7 filters.

6.2.1. Overall accuracy

Table 15 reports the overall model accuracies on the DCASE2019 dataset. The floating point models SB-CNN, SB-CNN-DS, Stride and Stride-DS model scores respectively +6.7%, +1.6%, +2.1% and -0.7% compared to the DCASE2019 baseline model while having respectively 91.8%, 22.0%, 82.5% and 39.5% of the parameters. Experiment RQ2-2 scores 5.9 to 7.4% lower than the DCASE2019 baseline model. The RQ2-3 results are poor and below the floating point reference variants. Experiment RQ2-4 yields a higher accuracy where the SB-CNN, SB-CNN-DS, Stride and Stride-DS model score respectively +0.7%, +0.5%, +1.4% and 1.1% higher when compared to no pretrained parameters. Due to the increased filter size of the models in RQ2-5, the number of weigths of the SB-CNN, SB-CNN-DS, Stride and Stride-DS model are respectively 1.3x, 1.0x, 1.4x and 0.5x the number of parameters of the baseline models. The SB-CNN, SB-CNN-DS, Stride and Stride-DS model scores respectively +2.7%, +1.6%, +2.5% and +0.5% higher than the baseline model variants. In experiment RQ2-6, the number of weigths of the SB-CNN, SB-CNN-DS, Stride and Stride-DS model are respectively 1.4x, 1.5x, 1.5x and 0.7x the number of parameters of the baseline models. The SB-CNN, SB-CNN-DS, Stride and Stride-DS model scores respectively +3.7%, +2.8%, +4.2% and +1.8% higher than the baseline model variants.

Experiment	RQ2-1	RQ2-2	RQ2-3	RQ2-4	RQ2-5	RQ2-6
Model	floating point	3 conv frozen	2 conv frozen	1 conv frozen	filter 7x7	filter 7x7 D100
SBCNN	**69.2%** ±0.3	55.1% ±0.3	66.8% ±0.5	**69.9%** ±0.2	71.9% ±0.7	72.9% ±0.3
SBCNN-DS	**64.1%** ±0.1	54.4% ±0.4	61.1% ±0.5	**64.6%** ±0.3	65.7% ±0.9	66.9% ±0.7
Stride	**64.6%** ±0.2	56.6% ±0.3	64.2% ±0.5	**66.0%** ±0.3	67.1% ±0.4	68.8% ±0.4
Stride-DS	**61.8%** ±0.3	56.6% ±0.5	61.4% ±0.2	**62.9%** ±0.2	62.3% ±0.8	63.6% ±0.5

Table 9: Overall accuracy results for DCASE 2019 development data set

6.2.2. Detailed accuracy

Table 10 illustrates the detailed accuracy per class for the highest scoring models using the baseline models. The highest accuracy increases are found in the airport (+17 to 23%), shopping mail (+10 to +16%) and bus (+9 to +23%) scene. The street pedestrian and metro scene scores overall lower. The latter two scenes also perform poorly on the 7x7 filter model variant and the model with extended dense layer. The detailed results of all experiments can be found in the appendix (section 8.2).

Model Test Accuracy - First convolution layer frozen					
Scene label	DCase Baseline	SB-CNN	SB-CNN-DS	Stride	Stride-DS
airport	48.4%	72.0%	72.7%	69.0%	65.5%
shopping mall	59.4%	69.1%	67.2%	70.1%	67.7%
metro station	54.5%	56.3%	51.4%	51.3%	46.9%
pedestrian street	60.9%	53.2%	49.1%	53.4%	51.9%
public square	40.7%	47.6%	41.0%	43.5%	38.0%
street traffic	86.7%	87.8%	85.2%	84.6%	84.4%
tram	64.0%	70.0%	59.3%	63.2%	60.7%
bus	62.3%	85.3%	75.5%	77.1%	72.5%
metro	65.1%	64.0%	52.5%	54.9%	51.0%
park	83.1%	93.4%	91.9%	93.3%	90.0%
Overall	**62.5%**	**69.9%**	**64.6%**	**66.0%**	**62.9%**

Table 10: RQ2-3 Accuracy results for DCASE 2019 development data set

6.2.3. Discussion

The RQ2 experiments investigate the effect on accuracy when networks designed for audio event classification are used in the context of audio scene classification. The reference is the DCASE2019 baseline model with 62.5% accuracy on the DCASE2019 evaluation dataset. The results of the experiments in context of RQ2 are reported in section 6.2. The experiments RQ2-1 to RQ2-3 evaluate the partial reuse of learned parameters of models trained on the UrbanSound8k dataset. Experiment RQ2-4 starts completely with random values. In experiment RQ2-5 and RQ2-6, the models inherit some properties from the DCASE2019 baseline model.

Experiment RQ2-1 is based on random parameters and achieves a higher score for the model models SBCNN, SBCNN-DS and Stride than the DCASE2019 baseline model. The reuse of convolution layers trained on the UrbanSound8k dataset is not always beneficial for accuracy. The worst result is achieved in RQ2-2 where three convolution layers are reused. All models fluctuate between around 55%. The RQ2-3 experiment uses two convolution layers. The SBCNN (-DS) models score 2.4% and 3% less than RQ2-1, respectively. The Stride and Stride-DS both score 0.4% lower than RQ2-1. The reuse of only the first convolution layer is advantageous for accuracy. The Stride models score 1.1 to 1.4% more than double SBCNN models with 0.5 to 0.7% accuracy gain.

Experiment RQ2-5 uses a 7x7 filter instead of a 5x5 filter for the first two convolution layers. The third convolution layer can not use a 7x7 filter as it would be larger than its input tensor. The depthwise separable convolution models achieve less gain from the filter size with 0.5-1.6% compared to the standard convolution models with 2.5-2.7% higher accuracy. In addition to the filter, experiment RQ2-6 also increased the dense layer from 64 neurons to 100 neurons. The depthwise separable convolution models again achieve less gain from the filter size with 1.8-2.8% compared to the standard convolution models with 3.7-4.2% higher accuracy. Remarkable is the SBCNN-DS model, which achieves 4.4% more accuracy than the DCASE2019 baseline model while having 3.1x fewer parameters.

The training hyperparameters (section 5.3) are identical for all reported results. Small experiments on the SB-CNN model have indicated that lower learning rates achieved a slightly higher accuracy on the DCASE2019 dataset. This might indicate that the models

could capture the characteristics of the audio scene sounds better by smaller weight updates. However, a learning rate of 0.001 increased the training time to 20h which makes it not computationally not feasible within the timeframe of the research.

6.2.4. Conclusion

The acoustic event models can classify audio scenes with an acceptable accuracy. The event models are capable of extracting useful features using a limited 0.72s audio input window compared to the 10s input window of the DCASE2019 baseline model. Experiment RQ2-1 has shown that the AEC networks SB-CNN, SB-CNN-DS, Stride achieve higher accuracy than the ASC DCASE2019 baseline model. The transferred parameters of the UrbanSound8k trained models are not beneficial when more than one layer is transferred. This might indicate that the features extracted by the first layer are general while the features of the middle and last convolution layer are specific for the UrbanSound8k dataset. Even though the UrbanSound8k is 10.1 times smaller than the DCASE2019 dataset, the weight transfer still provides an accuracy gain of 0.5 to 1.4%. The accuracy gain for depthwise separable convolution models was twice the gain of classic convolution. The model improvement using the 7x7 filter size and increased dense layer are beneficial for the DCASE2019 scene classification.

The overall effect on the DCASE2019 accuracy is positive for the SB-CNN, SB-CNN-DS and Stride model in experiments RQ2-1, 4, 5 and 6. The Stride-DS model does not score significantly better in any experiment. However, all thesis models score very poorly on the pedestrian street and metro classes compared to the DCASE2019 baseline model accuracy (appendix). This indicates that the input size remains critical for these audio scenes. An explorative test of doubling the input size of the thesis models to 60x62 achieves an increased overall accuracy of 75.8%, 70.2%, 73.5% and 69.3% respectively for the SB-CNN, SB-CNN-DS, Stride and Stride-DS model. In this configuration, the SB-CNN model scores 0.5% and 7.1% better on the pedestrian street and metro class compared to the DCASE2019 baseline model (appendix section 8.2.7).

6.3. Results RQ3

The third research question is related to the impact of an implantable MEMS microphone. Depending on the hardware and user application, the recording characteristics of the audio for deployed network models can be different than the audio used during training. RQ3 investigates three different input features in this condition: ***Which input feature representation among Short-Time Fourier Transform, Mel spectrogram, Constant Q spectrogram achieves the highest Urbansound8k accuracy on the SB-CNN(-DS) and Stride(-DS) models using an implanted MEMS microphone?***

The RQ3 experiments investigate the effect on accuracy when the UrbanSound8k dataset is represented using data features STFT, CQT and Mel. The thesis models are trained using the original UrbanSound8k sound clips with augmentation. Next, the models are tested with the same sound clips but recorded through a subcutaneous microphone according to Calero et al. [2018]. The clips are filtered using a 1th order Butterworth BPF 100Hz-8kHz and 200Hz-8kHz band pass filter to simulate the frequency sensitivity of the microphone. The attenuation on the cut-off frequency is 3dB. The audio information outside the pass band will be reduced with 6dB per octave. Because the amplitude information is reduced below and above the pass band, the accuracy loss cannot be clearly assigned to the lower

or higher frequency information loss. Therefore the HPF 100Hz, HPF 200Hz and LPF 8kHz are tested as well.

6.3.1. STFT

The short-term Fourier transformation is extracted using the librosa python library using the librosa.core.stft function. The phase output of the 1024 point FFT has been discarded. The magnitude output has been grouped by the numpy function split_array() to divide 513 bins into 60 groups to match the model input tensor. The function returns length modulo n sub-arrays of size length/n+1 and the rest of size length/n. This results in the first 33 groups having nine bins and the rest containing eight bins. All bins are summed into a final group result. Four train sessions have been executed per models. Table 11 reports the results using the 95% confidence interval.

Experiment	STFT	BPF 100-8kHz	BPF 200-8kHz	LPF 8kHz	HPF 100Hz	HPF 200Hz
SBCNN	**66.7** ±0.5	48.6 ±1.0	**46.2** ±0.8	51.7 ±1.0	64.8 ±0.7	61.3 ±0.7
SBCNN-DS	**65.8** ±0.5	44.2 ±1.1	**41.5** ±1.0	46.6 ±1.0	63.8 ±0.7	60.3 ±0.5
Stride	**66.6** ±0.3	50.6 ±0.9	**47.3** ±0.7	53.6 ±1.0	64.3 ±0.4	60.6 ±0.5
Stride-DS	**65.0** ±0.6	46.7 ±0.7	**43.9** ±0.8	50.0 ±0.7	63.1 ±0.8	60.0 ±0.6

Table 11: Overall accuracy results for Urbansound8k STFT based feature

The SBCNN, SBCNN-DS, Stride, and Stride-DS-24 score respectively 4.7%, 4.3%, 3.5% and 4% lower using the STFT feature compared to the Mel spectrogram. The band pass filtered signal between 200Hz and 8kHz scores the lowest of the audio signals with respectively 20.5%, 24.3%, 19.3% and 21.1% less accuracy than the audio signals used during training. The average accuracy drop for the HPF 100Hz, HPF 200Hz and LPF 8kHz is respectively 2.3%, 5.5% and 15.6%.

6.3.2. CQT

Next to the sample rate (22050Hz) and the number of output bins (60), the function also requires the bins per octave information. The output bins are grouped into a set of octaves relative to the minimum frequency. By defining the number of bins per octave (BPO), the frequency of the latest bin is defined. When the number of bins per octave is high, the number of octaves decreases and the frequency range of the transformation might not capture the available frequency range of the signal. Therefore a set of CQT transformations has been generated to maximize the overlap with the available signal information up until the nyquist frequency (11025Hz) using different frequency spacings. Eight CQT configurations have been created by changing the bins per octave from 6 to 13 using start frequencies 11Hz, 28Hz, 60Hz, 107Hz, 169Hz, 250Hz, 341Hz and 444Hz respectively. Four train sessions have been executed per model for all BPO configurations. Table 12 reports the results using the 95% confidence interval.

Model Test Accuracy CQT								
Model	BPO6	BPO7	BPO8	BPO9	BPO10	BPO11	BPO12	BPO13
SbCnn	**65.4**±1.4	67.5±1.0	67.3±1.7	**68.7**±0.7	68.6±1.0	67.4±1.5	67.8±1.1	66.4±0.9
SbCnnDs	**63.4**±1.9	65.6±2.0	66.4±0.6	**68.2**±0.7	67.9±1.0	66.7±1.8	66.5±2.4	64.4±0.2
Stride	**65.2**±1.2	66.1±0.8	67.1±1.1	**68.3**±1.7	68.0±1.1	67.3±1.3	67.0±1.1	65.0±1.1
StrideDs24	**63.5**±0.7	64.6±0.4	66.0±1.1	**66.8**±1.4	66.1±0.8	64.5±1.6	64.5±0.7	62.7±1.3

Table 12: Accuracy results for CQT on Urbansound8k data set

The best scoring configuration uses nine bins per octave. The difference between the classic convolution and depthwise separable convolution ranges from 0.5% (BPO9) to 2% for SBCNN and 1.1% (BPO8) to 2.8% for the Stride model. The SBCNN, SBCNN-DS, Stride, and Stride-DS-24 score respectively 2.7%, 1.9%, 1.8% and 2.2% lower using the CQT feature compared to the Mel spectrogram.

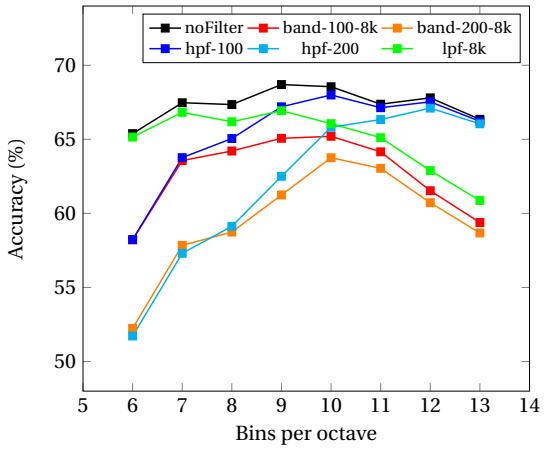

Figure 34: SB-CNN model CQT accuracy

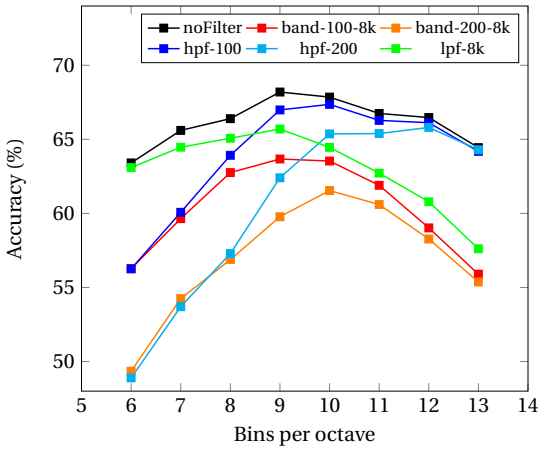

Figure 35: SB-CNN-DS model CQT accuracy

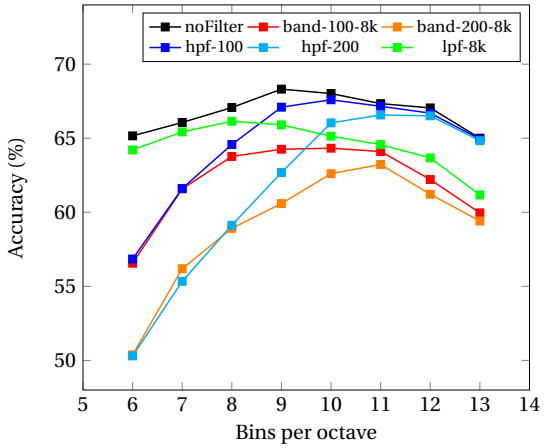

Figure 36: Stride model CQT accuracy

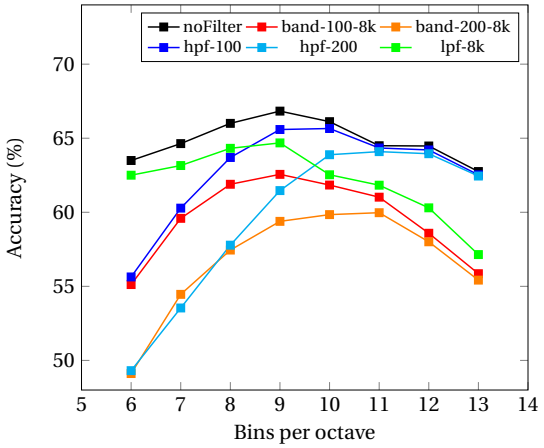

Figure 37: Stride-DS model CQT accuracy

The SBCNN, SBCNN-DS, Stride, and Stride-DS-24 is tested band pass filtered audio between 200Hz and 8kHz, the accuracy drop is respectively 7.46%, 8.42%, 7.72% and 7.44% for BPO9, 4.8%, 6.3%, 4.1% and 4.5% for BPO10 and 4.32%, 6.15%, 4.11% and 4.52% for BPO11.

The average accuracy drop for the HPF 100Hz, HPF 200Hz and LPF 8kHz is respectively for BPO9 1.29%, 5.74%, 2.20% for BPO10 0.48%, 2.35%, 3.08% and 0.26%, 0.89%, 2.93% for BPO11.

6.3.3. MEL SPECTROGRAM

The results with non-filtered audio are the floating point results of RQ1. Ten train sessions have been executed per model. Table 13 reports the results using the 95% confidence interval.

Experiment	Mel	BPF 100-8kHz	BPF 200-8kHz	LPF 8k	HPF 100	HPF 200
SBCNN	**71.4** ±0.8	68.6 ±0.7	**65.0** ±0.6	70.6 ±0.8	69.5 ±0.6	65.5 ±0.6
SBCNN-DS	**70.1** ±0.6	67.3 ±0.7	**63.1** ±0.5	69.3 ±0.9	68.3 ±0.8	63.9 ±0.8
Stride	**70.1** ±0.5	66.6 ±0.5	**63.2** ±0.6	68.8 ±0.5	68.5 ±0.4	64.8 ±0.7
Stride-DS	**69.0** ±0.6	65.7 ±0.5	**61.9** ±0.7	67.9 ±0.4	67.1 ±0.6	63.4 ±0.6

Table 13: Overall accuracy results for Urbansound8k Mel based feature

The SBCNN, SBCNN-DS, Stride, and Stride-DS-24 score respectively 6.4%, 7%, 6.9% and 7.1% lower using band pass filtered audio between 200Hz and 8kHz. The average accuracy drop for the HPF 100Hz, HPF 200Hz and LPF 8kHz is respectively 3.05%, 7% and 2.25%.

6.3.4. OVERALL COMPARISON

Table 14 reports the average accuracy score of the SBCNN, SBCNN-DS, Stride, and Stride-DS-24 models of the STFT, CQT and Mel feature.

Experiment	No filter	BPF 100-8kHz	BPF 200-8kHz	LPF 8k	HPF 100	HPF 200
STFT	**66.0%**	47.5%	**44.7%**	50.5%	63.8%	60.6%
CQT BPO6	**64.4%**	56.6%	**50.3%**	63.7%	56.7%	50.1%
Mel	**70.2%**	67.1%	**63.3%**	69.2%	68.4%	64.4%

Table 14: Overall accuracy results for Urbansound8k

6.3.5. EXPLORATORY EXPERIMENT

In previous experiments, the thesis models were trained using original UrbanSound8k sounds and tested using band pass filtered sounds. This exploratory experiment trained the SB-CNN model on band pass filtered sounds and tested the accuracy using non-filtered sounds. The experiment was executed only once.

	Trained BPF 100-8kHz		Trained BPF 200-8kHz	
Feature	Tested BPF 100-8kHz	Tested No filter	Tested BPF 200-8kHz	Tested No filter
STFT	68.2%	53.4%	67.7%	50.3%
CQT BPO6	67.3%	56.6%	66.9%	50.6%
Mel	73.1%	68.4%	72.1%	60.9%

Table 15: Urbansound8k accuracy results for SB-CNN model trained on BPF data

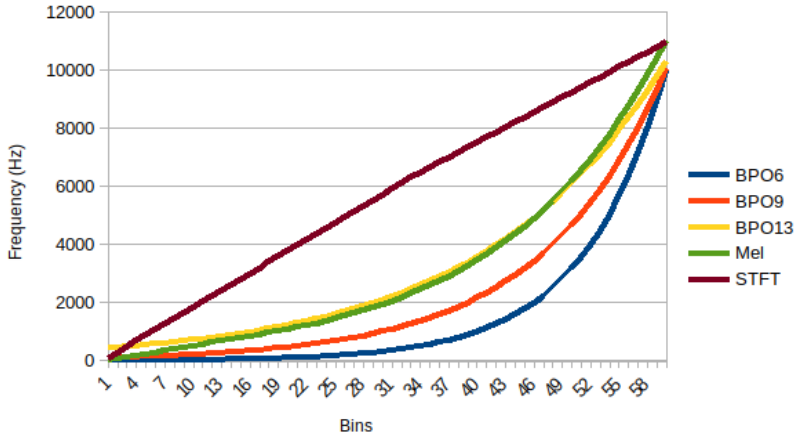

Figure 38: Bin Frequency relation of STFT, CQT and Mel data feature

6.3.6. Discussion

The average model accuracy of the STFT feature scores 4.1% lower compared to the average model accuracy of the Mel feature. The accuracy drops significantly when the models are tested with band pass filtered signal. The average model accuracy loss is 18.5 and 21.5% for respectively 100Hz-8kHz and 200Hz-8kHz. Most accuracy is lost on the high-end side of the BPF with an average model accuracy loss of 15.5% for frequencies above 8kHz.

The best CQT scoring configuration is the BPO10 with an average model loss of 2.46% compared to Mel. However a more fair comparison with STFT and Mel is configuration BPO6. The center frequency of the first filter in the BPO6 configuration is 11Hz which comes closest to the spectral coverage of the STFT and Mel features. The average CQT BPO6 feature score is 5.7% lower compared to the Mel feature. The band pass filtered audio drops the average model accuracy with 7.9 and 14.1% for respectively 100Hz-8kHz and 200Hz-8kHz. Most accuracy loss is found at the low-end side of the BPF with 7.7% and 14.3% for HPF 100Hz and HPF 200Hz respectively. The least model average accuracy loss of 0.6% is at the high frequencies above 8kHz.

The Mel feature score shows an accuracy drop of 3.1 to 6.9% on the band pass 100Hz-8kHz and 200Hz-8kHz respectively. Most average model accuracy loss is located in the low-frequency information with 1.8% and 5.8% for HPF 100Hz and HPF 200Hz respectively. The least average model accuracy loss of 0.95% is found at the high frequencies above 8kHz.

The Mel feature representation achieves the highest accuracy on the original UrbanSound8k sounds. The STFT scores 4.2% less using an equal amount of bins over the frequency range. The CQT BPO6 scores 5.8% less using much more low-frequency resolution with 41 bins below 1kHz (compared to 18 bins for Mel and six bins for STFT). When the number of bins increases, fewer octaves are used to cover the 60 input bins. When the end frequency is fixed, the start frequency needs to increase. The consequence is that the octaves will grow faster in frequency. The shape of the CQT becomes less steep and matches more with the Mel shape (Figure 38). Even though the BPO13 is similar to the Mel shape, the CQT feature scores less because the start frequency is already at 444Hz. The Mel feature does not miss the low-frequency information and captures more information as it starts from 0Hz and is linear until 1kHz.

When the UrbanSound8k sounds are band pass filtered, the accuracy drops for all data

features. The high-end attenuation above 8kHz affects the STFT feature more as it drops 15.5% over 18 STFT bins while the CQT and Mel drop only 0.7 to 1% spanning respectively 3 and 6 bins above 8kHz. The low-end attenuation below 200Hz affects the CQT feature more as it spans 27 CQT bins with an accuracy loss of 14.3%. Even though less bins are affected in the low-end attenuation for STFT (2 bins) and Mel (5 bins) the loss of respectively 5.4% and 5.8% is larger than the loss at the high-end attenuation. This might indicate that the low frequency information is more important in the Urbansound8k classification. The Mel feature has the best of both worlds with almost no accuracy loss at the high-end attenuation and a relative low accuracy loss at the low-end attenuation.

6.3.7. CONCLUSION

The Mel feature representation achieves the highest UrbanSound8k accuracy using simulated MEMS microphone audio. It outperforms the CQT and STFT data feature on filtered data as well the original data. The experiments have indicates that the CQT can perform better by optimizing the start frequency and the number of bins per octave. However, the frequency range of the STFT and Mel should cover the same range for fair comparison.

The accuracy of all models dropped significantly when the test sounds were filtered using a 200Hz-8kHz band pass of 6dB attenuation per octave. An exploratory experiment has indicated that a similar effect exists in a reversed scenario as well. When the SB-CNN model was trained on band filtered data, the accuracy score drops significantly when tested on the original audio. Therefore it is recommended to consider the microphone frequency sensitivity of the target device during the training process to avoid accuracy loss during the deployment phase.

7. CONCLUSIONS

This thesis has investigated to which extend CNNs can be used on tiny embedded devices in context of audio classification. The research was focused on the accuracy of CNNs when deployed on implantable hearing devices, facing three challenges.

The first challenge was related to the hardware resource constraints. The research investigated the effect of quantization on four convolutional neural networks, designed for the low power microcontroller STM32L476 using the UrbanSound8k dataset. The arithmetic precision has been reduced from 32bit floating point to 8bit integer precision using post quantization and training aware quantization techniques. The accuracy loss was limited from 0 to 0.3% while reducing the model memory size 3.5 times. The models performed inferences 2.6 times faster. Although quantization has been successfully applied to well parameterized neural networks, research work (Jacob et al. [2017]) also indicated that post quantization might not work well on small models. The experiments have demonstrated that post-quantization performs better or equal than quantization aware training on the UrbanSound8k dataset. Therefore post quantization method might be worth considering as the first quantization option as it does not require model retraining and allow reuse of existing models without the original dataset.

The second challenge was associated with network model type and the nature of the sounds that need to be classified. This research indicated that it is feasible to use an acoustic event classification model on an acoustic scene classification task. Experiments using transfer learning have increased the accuracy of all thesis models above the DCASE2019 baseline accuracy. Model architecture improvements regarding filter size and dense layer increased the overall accuracy. However, all experiments performed poorly on the street pedestrian and metro class, demonstrating a trade-off between minimum input size and accuracy.

The last challenge was linked to the microphone frequency response of the application hardware. Experiments evaluated the STFT, CQT and Mel data feature when presented with simulated sounds of the implanted microphone (Calero et al. [2018]). The Mel feature achieved the highest accuracy on the thesis models among the STFT and CQT features. The results have shown significant accuracy loss on all models and feature combinations when the test audio is not recorded with the same frequency sensitivity as the training data. Further investigations on the impact of deployment versus training recording conditions are required.

The depthwise separable convolution algorithm is preferred over the classic convolution algorithm in terms of inference speed. The mathematical optimization reduced the number of multiply and accumulate operations by a factor of five which demonstrated three times faster models. If the models are already small, decreasing the number of learnable parameters might result in too few parameters to properly learn during training. The accuracy loss depends on the task or dataset at hand. On all UrbanSound8k experiments, the accuracy loss was up to 2% while the DCASE2019 scene classification lost up to 5% accuracy.

The 8bit quantization has a negligible accuracy impact on the UrbanSound8k classification using depthwise separable convolution. The combination of quantization and depthwise separable convolution resulted in a model size reductions up to 10 times with only 1.5% accuracy loss. This creates opportunities to drive quantization further to lower bitwidth representations. However, it is unknown at which point the depthwise separable

convolution will be impacted nor if this is the right convolution type going forward in audio classification.

The Stride-DS model is the most advantageous model for an implantable hearing application. The model is suitable for event and scene classification requiring only 40ms inference time. The gain of battery autonomy is more beneficial versus the accuracy loss compared to the other models. The model memory of 68kB, including both RAM and flash parameters, provides space for application and audio preprocessing code with the predefined 128kB limit.

The models have not been tested on real-world audio. Although the classification accuracy has been measured on the dataset, it does not indicate what will be the prediction output for sounds that do not belong to any of the classification classes. More research is required to evaluate the models in a realistic scenario using open set classification or another algorithm that can distinguish between the known and unknown data.

From a wide range of experiments, this research shows that convolutional neural networks for audio classification can be effectively reduced in size to make it suitable for tiny embedded devices. However, the edge hardware specifications must be taken into account. The frequency-specific output of an embedded microphone can lead to significant accuracy loss during deployment.

7.1. Future work

The experiments of RQ3 have indicated a significant drop in accuracy when a model is trained with original data and tested with bandpass filtered data using three different data features. When less information becomes available, the network might miss key information. An exploratory test has suggested that the opposite is also true. If the network is trained with filtered data, the accuracy also drops significantly when more information is provided to the network. More research is required to understand this phenomenon and keep an acceptable application accuracy in deployment stage. The spectrum data augmentation techniques for audio classification compared by Wei et al. [2020] might be a good starting point.

The power consumption can be further improved by quantizing to lower bit widths. Recently a new quantization technique, called Quantization Guided Training, was introduced by (Ghamari et al. [2021]). The technique can be applied to hybrid models with mixed bit precision down to 2-bit. Nowdays semiconductor manufacturers (Syntiant [2020], ARM [2020]) are producing neural processing units (NPU) with dedicated arithmetic logic to execute neural network algorithms faster and more energy-efficient. The hardware provides a platform to evaluate more aggressive quantization using bit widths of 1-, 2-, 4- and 8bit weights.

The literature does not provide a clear answer on how the representative dataset affects the quantization quality. The post quantization process uses a representative dataset to calibrate the post quantization scaling. The representative dataset is a set of input data values that are large enough to represent typical values. If the data is too small, the scaling might clip the model during deployment. If the data is too large, the scaling will generate large rounding errors. In both cases, the accuracy is affected negatively. Another optimization for quantization quality is the removal of outliers in the weight value distribution which can improve the quantization precision for all weight values.

The models have not been evaluated with data from unknown classes. More research is

required to test the models in a realistic scenario using open set classification (Geng et al. [2018]) or another algorithm that can distinguish between the known and unknown data. The algorithm could be tested on SONYC Urban Sound Tagging dataset of Mark Cartwright and Bello [2019] which contains sounds related to UrbanSound8k classes and unknown classes.

8. Appendix

8.1. RQ1 experiments

8.1.1. SB-CNN(-DS)

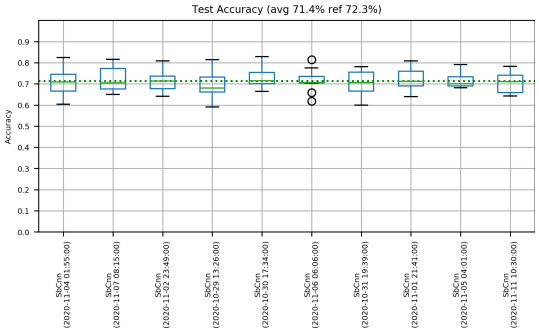

Figure 39: Test accuracy SB-CNN float model

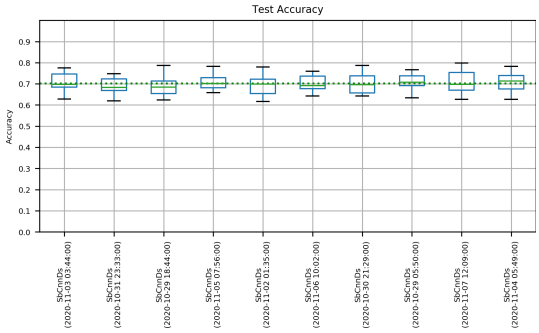

Figure 40: Test accuracy SB-CNN-DS float model

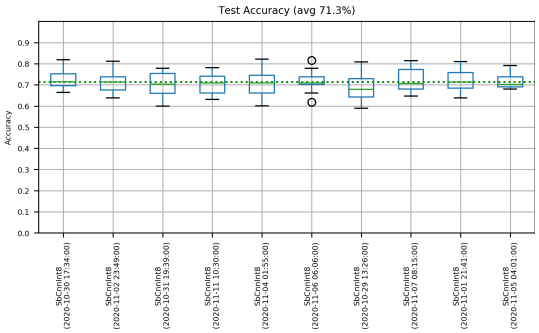

Figure 41: Test accuracy SB-CNN post quantization model

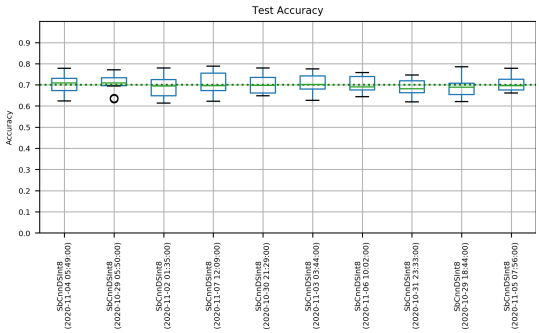

Figure 42: Test accuracy SB-CNN-DS post quantization model

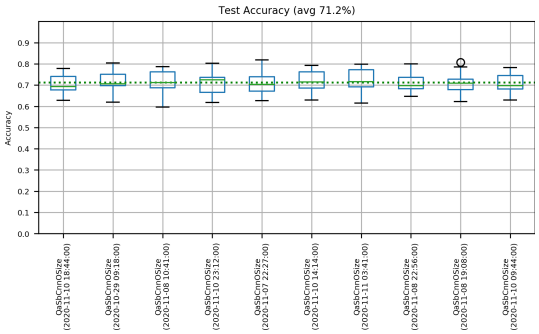

Figure 43: Test accuracy SB-CNN quantization aware model

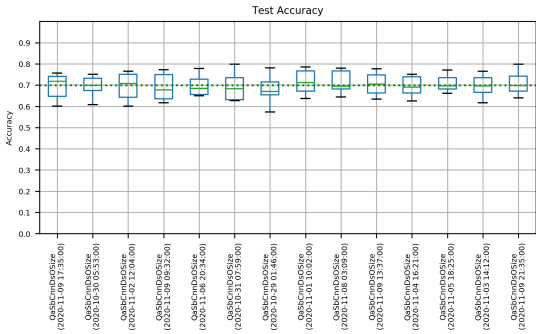

Figure 44: Test accuracy SB-CNN-DS quantization aware model

8.1.2. STRIDE(-DS)

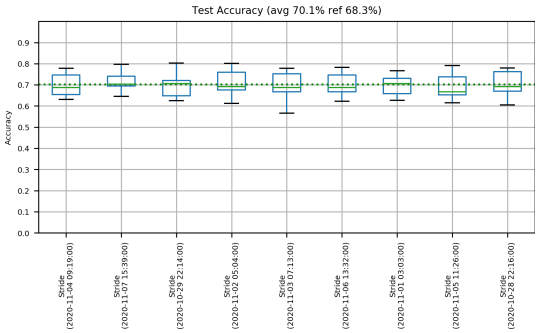

Figure 45: Test accuracy Stride float model

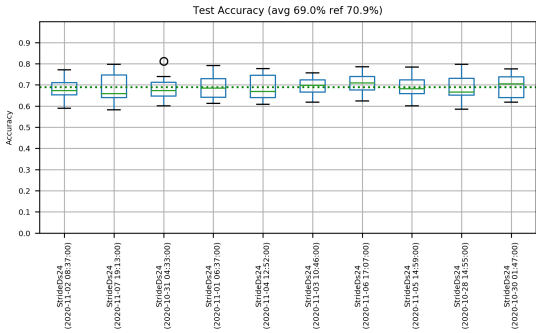

Figure 46: Test accuracy Stride-DS float model

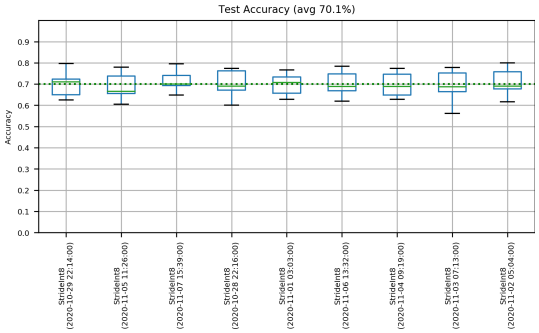

Figure 47: Test accuracy Stride post quantization model

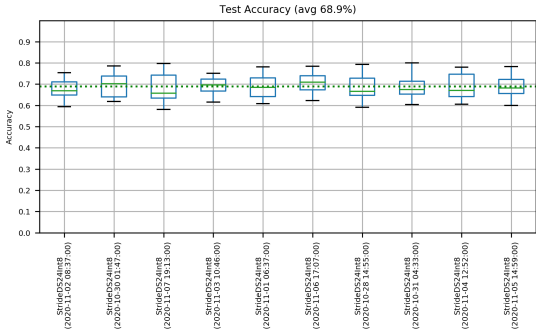

Figure 48: Test accuracy Stride-DS post quantization model

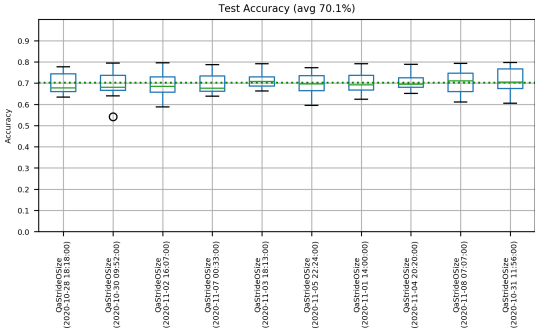

Figure 49: Test accuracy Stride quantization aware model

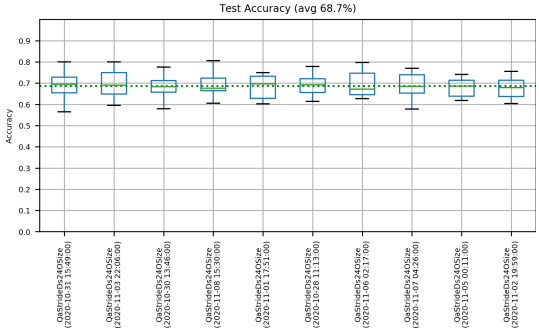

Figure 50: Test accuracy Stride-DS quantization aware model

8.2. RQ2 Experiments

8.2.1. Experiment RQ2-1

Model Test Accuracy - Full retrain					
Scene label	DCase Baseline	SB-CNN	SB-CNN-DS	Stride	Stride-DS
airport	48.4%	69.3%	67.9%	63.9%	62.5%
shopping mall	59.4%	70.2%	64.6%	70.5%	67.5%
metro station	54.5%	57.8%	51.2%	50.5%	44.5%
street pedestrian	60.9%	54.4%	49.8%	51.7%	49.1%
public square	40.7%	45.9%	41.4%	38.4%	37.0%
street traffic	86.7%	87.3%	84.8%	84.2%	83.1%
tram	64.0%	67.5%	62.2%	66.6%	60.9%
bus	62.3%	85.1%	76.5%	79.7%	76.5%
metro	65.1%	61.4%	51.6%	48.9%	46.9%
park	83.1%	93.3%	90.9%	91.8%	89.7%
Overall	62.5%	69.2%	64.1%	64.6%	61.8%

Table 16: Accuracy results for DCASE 2019 development data set

8.2.2. Experiment RQ2-2

Model Test Accuracy - 3 conv layers frozen					
Scene label	DCase Baseline	SB-CNN	SB-CNN-DS	Stride	Stride-DS
airport	48.4%	54.0%	54.1%	57.6%	57.1%
shopping mall	59.4%	58.6%	56.3%	59.8%	61.6%
metro station	54.5%	34.8%	39.3%	38.6%	35.8%
street pedestrian	60.9%	48.6%	44.5%	48.2%	48.7%
public square	40.7%	29.4%	30.3%	32.5%	33.3%
street traffic	86.7%	83.3%	81.2%	81.0%	81.4%
tram	64.0%	48.9%	47.3%	48.1%	47.9%
bus	62.3%	70.1%	67.0%	69.2%	68.6%
metro	65.1%	37.8%	37.0%	43.1%	46.5%
park	83.1%	85.7%	86.6%	87.7%	85.3%
Overall	62.5%	55.1%	54.4%	56.6%	56.6%

Table 17: Accuracy results for DCASE 2019 development data set

8.2.3. EXPERIMENT RQ2-3

Model Test Accuracy - 2 conv layers frozen					
Scene label	DCase Baseline	SB-CNN	SB-CNN-DS	Stride	Stride-DS
airport	48.4%	68.2%	62.2%	64.5%	63.1%
shopping mall	59.4%	68.9%	65.4%	69.2%	63.6%
metro station	54.5%	51.4%	43.8%	47.5%	43.7%
street pedestrian	60.9%	51.4%	51.5%	52.7%	53.2%
public square	40.7%	45.1%	37.6%	40.1%	36.3%
street traffic	86.7%	85.9%	83.5%	85.1%	82.4%
tram	64.0%	64.1%	53.0%	59.7%	62.2%
bus	62.3%	82.1%	71.8%	77.0%	70.0%
metro	65.1%	57.5%	50.6%	54.3%	50.1%
park	83.1%	93.4%	92.0%	91.8%	89.6%
Overall	62.5%	66.8%	61.1%	64.2%	61.4%

Table 18: Accuracy results for DCASE 2019 development data set

8.2.4. EXPERIMENT RQ2-4

Model Test Accuracy - 1 conv layers frozen					
Scene label	DCase Baseline	SB-CNN	SB-CNN-DS	Stride	Stride-DS
airport	48.4%	72.0%	72.7%	69.0%	65.5%
shopping mall	59.4%	69.1%	67.2%	70.1%	67.7%
metro station	54.5%	56.3%	51.4%	51.3%	46.9%
street pedestrian	60.9%	53.2%	49.1%	53.4%	51.9%
public square	40.7%	47.6%	41.0%	43.5%	38.0%
street traffic	86.7%	87.8%	85.2%	84.6%	84.4%
tram	64.0%	70.0%	59.3%	63.2%	60.7%
bus	62.3%	85.3%	75.5%	77.1%	72.5%
metro	65.1%	64.0%	52.5%	54.9%	51.0%
park	83.1%	93.4%	91.9%	93.3%	90.0%
Overall	62.5%	69.9%	64.6%	66.0%	62.9%

Table 19: Accuracy results for DCASE 2019 development data set

8.2.5. EXPERIMENT RQ2-5

Model Test Accuracy - 7x7 filter					
Scene label	DCase Baseline	SB-CNN	SB-CNN-DS	Stride	Stride-DS
airport	48.4%	71.8%	68.5%	70.1%	62.2%
shopping mall	59.4%	72.3%	68.7%	72.7%	71.1%
metro station	54.5%	64.4%	53.1%	53.6%	45.9%
street pedestrian	60.9%	55.8%	51.8%	52.8%	51.7%
public square	40.7%	48.1%	40.8%	41.7%	37.3%
street traffic	86.7%	88.6%	87.1%	84.8%	83.1%
tram	64.0%	73.4%	60.7%	67.6%	60.4%
bus	62.3%	85.9%	78.3%	81.0%	75.8%
metro	65.1%	65.3%	57.2%	53.8%	46.2%
park	83.1%	93.5%	91.2%	93.3%	89.2%
Overall	62.5%	71.9%	65.7%	67.1%	62.3%

Table 20: Accuracy results for DCASE 2019 development data set

8.2.6. EXPERIMENT RQ2-6

Model Test Accuracy - 7x7 filter - Dense 100					
Scene label	DCase Baseline	SB-CNN	SB-CNN-DS	Stride	Stride-DS
airport	48.4%	75.1%	71.9%	71.5%	66.4%
shopping mall	59.4%	74.2%	69.4%	73.0%	71.0%
metro station	54.5%	64.6%	55.7%	55.6%	48.1%
street pedestrian	60.9%	54.1%	51.0%	55.9%	49.9%
public square	40.7%	50.4%	42.6%	45.3%	39.7%
street traffic	86.7%	87.5%	86.4%	85.5%	84.4%
tram	64.0%	74.8%	63.1%	68.1%	59.4%
bus	62.3%	85.6%	81.4%	83.0%	77.8%
metro	65.1%	68.7%	55.2%	56.1%	48.9%
park	83.1%	94.5%	91.8%	93.5%	89.9%
Overall	62.5%	72.9%	66.9%	68.8%	63.6%

Table 21: Accuracy results for DCASE 2019 development data set

8.2.7. EXPLORATORY TEST

This section reports the exploratory testing using the thesis model architecture with increased input size of 60x62.

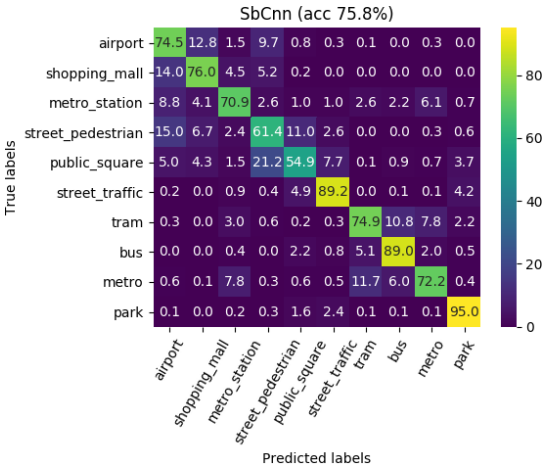

Figure 51: Confusion matrix SB-CNN model with 60x62 input

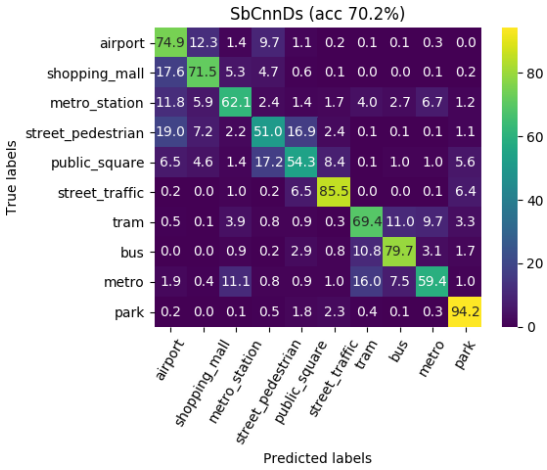

Figure 52: Confusion matrix SB-CNN-DS model with 60x62 input

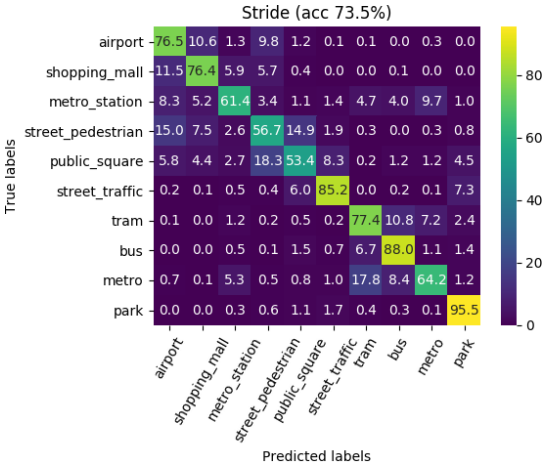

Figure 53: Confusion matrix Stride model with 60x62 input

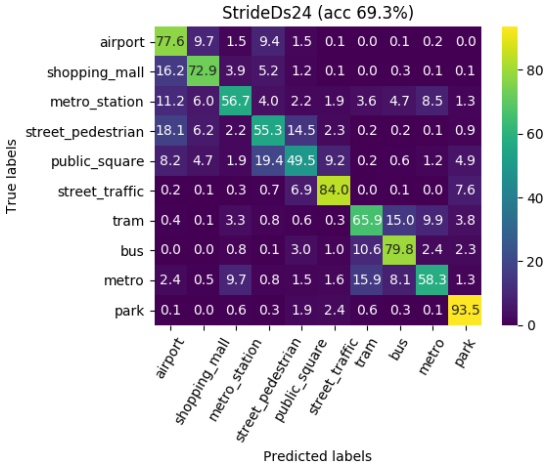

Figure 54: Confusion matrix Stride-DS model with 60x62 input

8.3. RQ3 EXPERIMENTS

8.3.1. EXPLORATORY TEST

This section reports the exploratory test by training the SB-CNN model on band pass filtered data while testing the model on non-filtered data.

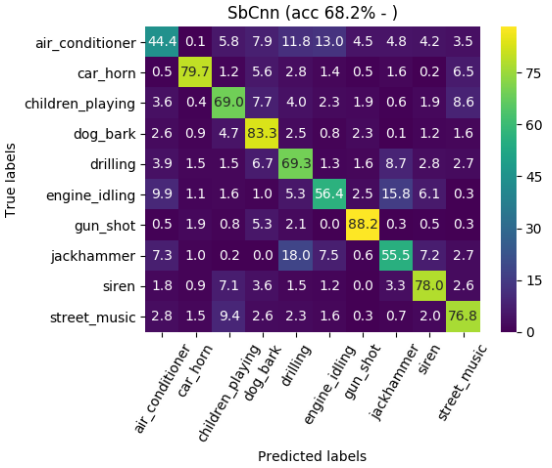

Figure 55: SB-CNN model trained and tested on STFT BPF 100Hz-8kHz

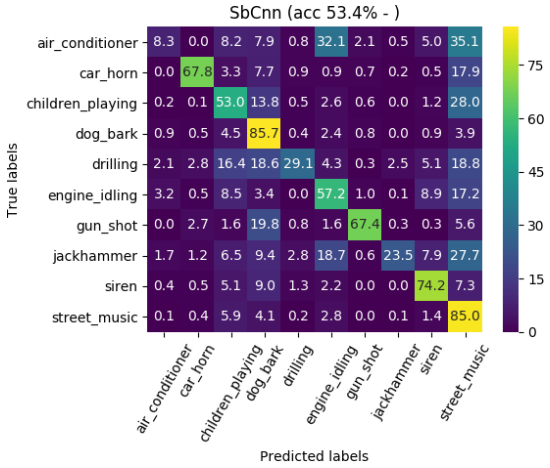

Figure 56: SB-CNN model trained on STFT BPF 100Hz-8kHz and tested on non-filtered data

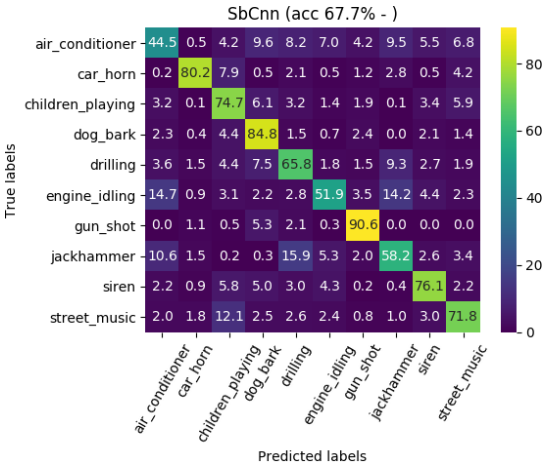

Figure 57: SB-CNN model trained and tested on STFT BPF 200Hz-8kHz

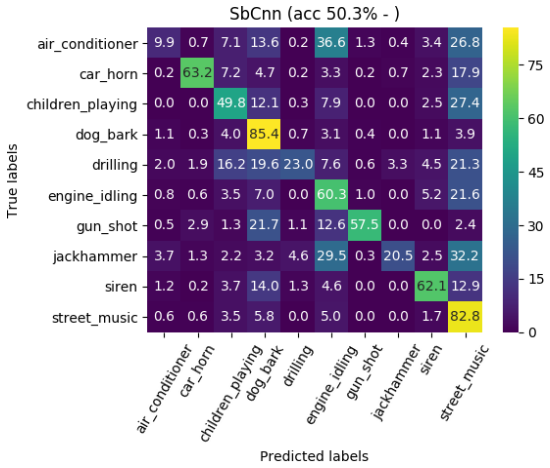

Figure 58: SB-CNN model trained on STFT BPF 200Hz-8kHz and tested on non-filtered data

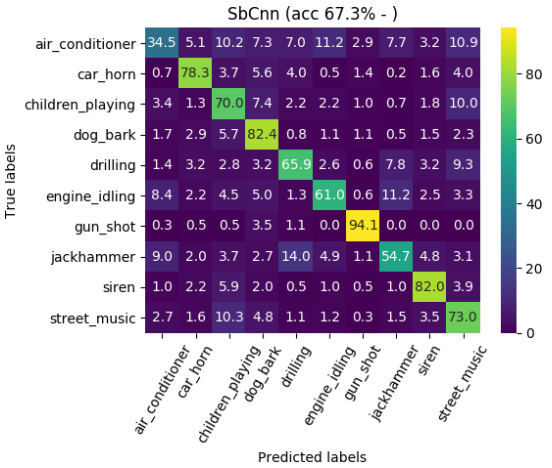

Figure 59: SB-CNN model trained and tested on CQT BPF 100Hz-8kHz

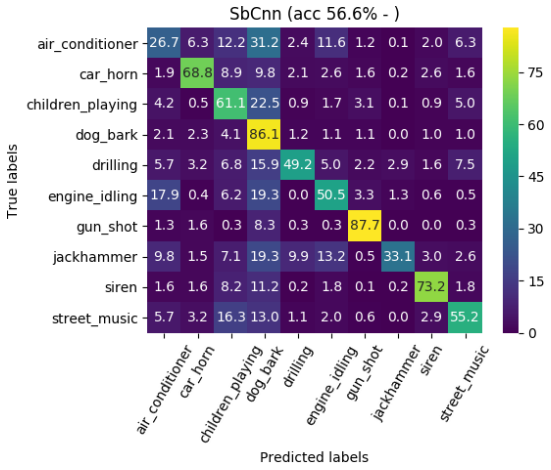

Figure 60: SB-CNN model trained on CQT BPF 100Hz-8kHz and tested on non-filtered data

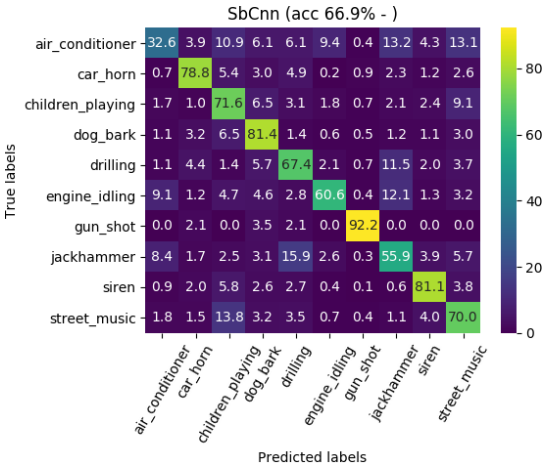

Figure 61: SB-CNN model trained and tested on CQT BPF 200Hz-8kHz

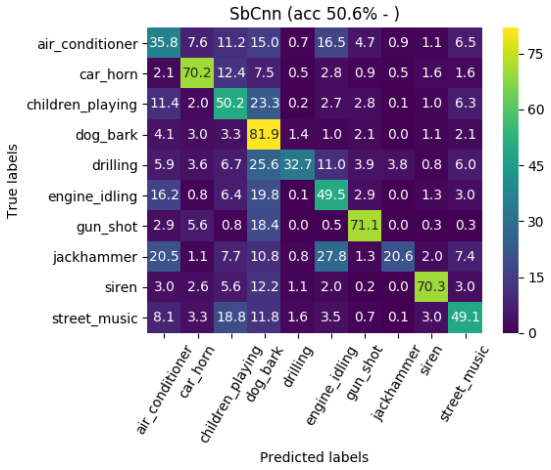

Figure 62: SB-CNN model trained on CQT BPF 200Hz-8kHz and tested on non-filtered data

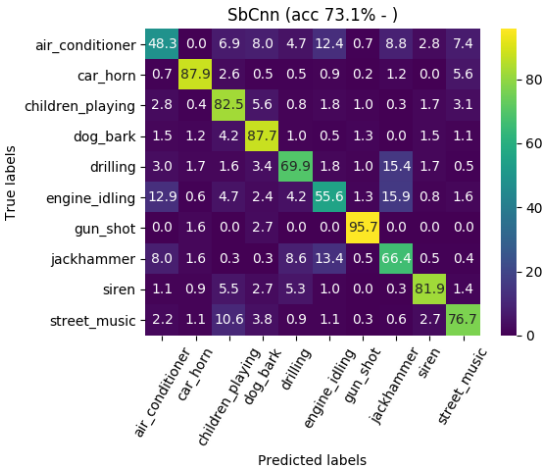

Figure 63: SB-CNN model trained and tested on Mel BPF 100Hz-8kHz

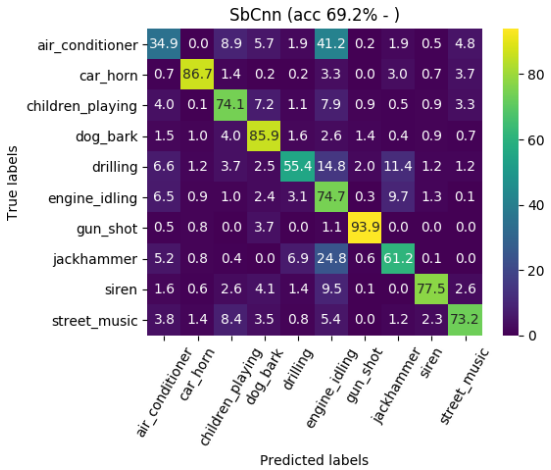

Figure 64: SB-CNN model trained on Mel BPF 100Hz-8kHz and tested on non-filtered data

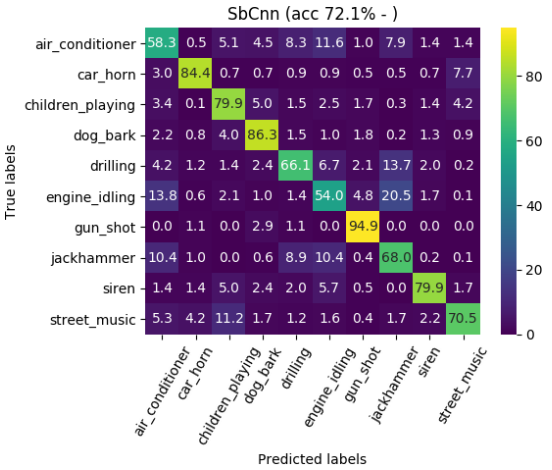

Figure 65: SB-CNN model trained and tested on Mel BPF 200Hz-8kHz

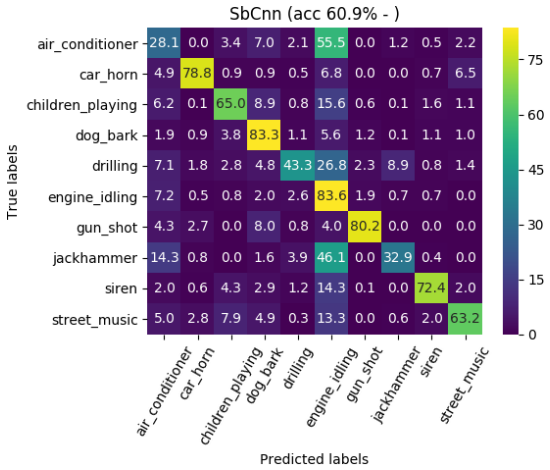

Figure 66: SB-CNN model trained on Mel BPF 200Hz-8kHz and tested on non-filtered data

BIBLIOGRAPHY

Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). URL http://data.europa.eu/eli/reg/2016/679/2016-05-04. 9

Github nordby jonnor. URL https://github.com/jonnor/ESC-CNN-microcontroller\commitf7a02189e18cc845dd4e912654a2509215475410. 42

Arm ai platform neural processor unit, 2020. URL https://www.arm.com/company/news/2020/10/latest-npu-adds-to-arm-ai-platform-performance. 63

Jakob Abeßer. A review of deep learning based methods for acoustic scene classification. *Applied Sciences*, 10(6), 2020. 8, 9

Cesare Alippi, Simone Disabato, and Manuel Roveri. Moving convolutional neural networks to embedded systems: The alexnet and vgg-16 case. pages 212–223, 04 2018. doi: 10.1109/IPSN.2018.00049. 21

J. Allen. Short term spectral analysis, synthesis, and modification by discrete fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(3):235–238, 1977. doi: 10.1109/TASSP.1977.1162950. 11

N. Aloysius and M. Geetha. A review on deep convolutional neural networks. In *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 0588–0592, 2017. doi: 10.1109/ICCSP.2017.8286426. 14

Laith Alzubaidi, Omran Al-Shamma, Mohammed Fadhel, and Laith Farhan. Optimizing the performance of breast cancer classification by employing the same domain transfer learning from hybrid deep convolutional neural network model. *Electronics*, 9:445, 03 2020. doi: 10.3390/electronics9030445. 4

Kamil Adiloglu; Jorg-Hendrik Bach. Hearing aid research data set for acoustic environment recognition (hear-ds). 2020. URL http://sigport.org/5392. 11

Jon Barker, Shinji Watanabe, Emmanuel Vincent, and Jan Trmal. The fifth 'chime' speech separation and recognition challenge: Dataset, task and baselines. 03 2018. 11

Judith C. Brown. Calculation of a constant q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434, 1991. doi: 10.1121/1.400476. 12

A. Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019. ISBN 9781999579517. URL https://books.google.be/books?id=0jbxwQEACAAJ. 15, 20

Diego Calero, Stephan Paul, André Gesing, Fabio Alves, and Júlio A. Cordioli. A technical review and evaluation of implantable sensors for hearing devices. *Biomedical engineering online*, 17(1):23–26, 2018. 31, 36, 56, 62

Hangting Chen, Zuozhen Liu, Zongming Liu, Pengyuan Zhang, and Yonghong Yan. Integrating the data augmentation scheme with various classifiers for acoustic scene modeling. Technical report, DCASE2019 Challenge, June 2019. 11, 12

François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016. URL http://arxiv.org/abs/1610.02357. 17

Yoni Choukroun, Eli Kravchik, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. *CoRR*, abs/1902.06822, 2019. URL http://arxiv.org/abs/1902.06822. 30

Heidi Christensen, Jon Barker, Ning Ma, and Phil Green. The chime corpus: a resource and a challenge for computational hearing in multisource environments. In *in Proc. Interspeech'10, Makuhari*, 2010. 10

N. Cristianini and B. Schölkopf. Support vector machines and kernel methods: The new generation of learning machines. *AI Mag.*, 23:31–42, 2002. 14

Gert Dekkers, Steven Lauwereins, Bart Thoen, Mulu Weldegebreal Adhana, Henk Brouckxon, Bertold Van den Bergh, Toon van Waterschoot, Bart Vanrumste, Marian Verhelst, and Peter Karsmakers. The sins database for detection of daily activities in a home environment using an acoustic sensor network. 2017. URL $$Uhttps://lirias.kuleuven.be/retrieve/525662$$D18-151.pdf[freelyavailable]. 10

John M. Eargle. *Robinson-Dadson Equal Loudness Contours*, pages 278–279. Springer US, Boston, MA, 2002. ISBN 978-1-4615-2027-6. doi: 10.1007/978-1-4615-2027-6_134. URL https://doi.org/10.1007/978-1-4615-2027-6_134. 7

Junxi Feng, Xiaohai He, Qizhi Teng, Chao Ren, Honggang Chen, and Yang Li. Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. *Physical Review E*, 100, 09 2019. doi: 10.1103/PhysRevE.100.033308. 16

P. Foster, S. Sigtia, S. Krstulovic, J. Barker, and M. D. Plumbley. Chime-home: A dataset for sound source recognition in a domestic environment. In *2015 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 1–5, 2015. doi: 10.1109/WASPAA.2015.7336899. 10

Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017. 10, 34

Chuanxing Geng, Sheng Jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *CoRR*, abs/1811.08581, 2018. URL http://arxiv.org/abs/1811.08581. 64

A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019. ISBN 9781492032618. URL https://books.google.be/books?id=HHetDwAAQBAJ. 16, 19

Sedigh Ghamari, Koray Ozcan, Thu Dinh, Andrey Melnikov, Juan Carvajal, Jan Ernst, and Sek Chai. Quantization-guided training for compact tinyml models, 2021. 63

I. J. Good. *Introduction to Cooley and Tukey (1965) An Algorithm for the Machine Calculation of Complex Fourier Series*, pages 201–216. Springer New York, New York, NY, 1997. ISBN 978-1-4612-0667-5. doi: 10.1007/978-1-4612-0667-5_9. URL https://doi.org/10.1007/978-1-4612-0667-5_9. 7

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: efficient inference engine on compressed deep neural network. *CoRR*, abs/1602.01528, 2016. URL http://arxiv.org/abs/1602.01528. 6

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 52

Toni Heittola, Annamaria Mesaros, and Tuomas Virtanen. Acoustic scene classification in dcase 2020 challenge: generalization across devices and low complexity solutions. pages 56–60, 2020. 9

Tin Kam Ho. Random decision forests. *Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995.*, page 278–282, 1995. 14

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017a. URL http://arxiv.org/abs/1704.04861. 21

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017b. URL http://arxiv.org/abs/1704.04861. 52

Khalid Hussain, Mazhar Hussain, and Muhammad Gufran Khan. Improved acoustic scene classification with dnn and cnn. *IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events (DCASE)*, 2017. 9

Muhammad Huzaifah. Comparison of time-frequency representations for environmental sound classification using convolutional neural networks. *CoRR*, abs/1706.07156, 2017. 31

Seo Hyeji and Park Jihwan. Acoustic scene classification using various pre-processed features and convolutional neural networks. Technical report, DCASE2019 Challenge, June 2019. 12

Keisuke Imoto. Introduction to acoustic event and scene analysis. *Acoustical Science and Technology*, 39, 05 2018. doi: 10.1250/ast.39.182. 7, 8, 10

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CoRR*, abs/1712.05877, 2017. URL http://arxiv.org/abs/1712.05877. 27, 28, 30, 31, 52, 62

Khaled Koutini, Hamid Eghbal-zadeh, and Gerhard Widmer. Acoustic scene classification and audio tagging with receptive-field-regularized CNNs. Technical report, DCASE2019 Challenge, June 2019. 12

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, abs/1806.08342, 2018. URL http://arxiv.org/abs/1806.08342. 27, 30, 52

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 22

Liangzhen Lai, Naveen Suda, and Vikas Chandra. Deep convolutional neural network inference with floating-point weights and fixed-point activations. *CoRR*, abs/1703.03073, 2017. URL http://arxiv.org/abs/1703.03073. 29

Jongpil Lee, Taejun Kim, Jiyoung Park, and Juhan Nam. Raw waveform-based audio classification using sample-level CNN architectures. *CoRR*, abs/1712.00866, 2017. URL http://arxiv.org/abs/1712.00866. 8

Darryl Dexu Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. *CoRR*, abs/1511.06393, 2015. URL http://arxiv.org/abs/1511.06393. 29

Graham Dove Jason Cramer Vincent Lostanlen Ho-Hsiang Wu Justin Salamon Oded Nov Mark Cartwright, Ana Elisa Mendez Mendez and Juan Pablo Bello. Sonyc urban sound tagging (sonyc-ust): a multilabel dataset from an urban acoustic sensor network, May 2019. URL https://doi.org/10.5281/zenodo.3233082. 64

Annamaria Mesaros, Toni Heittola, Aleksandr Diment, Benjamin Elizalde, Ankit Shah, Emmanuel Vincent, Bhiksha Raj, and Tuomas Virtanen. Dcase 2017 challenge setup: Tasks, datasets and baseline system. 11 2017a. 11

Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. Assessment of human and machine performance in acoustic scene classification: Dcase 2016 case study. pages 319–323, 10 2017b. doi: 10.1109/WASPAA.2017.8170047. 8

Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. A multi-device dataset for urban acoustic scene classification. *arXiv preprint arXiv:1807.09840*, 2018. 9

Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. Acoustic scene classification in dcase 2019 challenge: Closed and open set classification and data mismatch setups. 2019a. 37

Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. Acoustic scene classification in dcase 2019 challenge: Closed and open set classification and data mismatch setups. pages 164–168, 01 2019b. doi: 10.33682/m5kp-fa97. 9, 34

Satoshi Nakamura, K. Hiyane, F. Asano, Takanobu Nishiura, and T. Yamada. Acoustical sound database in real environments for sound scene understanding and hands-free speech recognition. *Proc. ICLRE*, pages 965–968, 01 2000. 10

Jon Nordby. Environmental sound classification on microcontrollers using convolutional neural networks, 2019. 22, 23, 31, 32, 33, 38, 41, 42, 50, 51

Soha Nossier, n Diaa, and Saleh Shehaby. Enhanced smart hearing aid using deep neural networks. *Alexandria Engineering Journal*, 58, 06 2019. doi: 10.1016/j.aej.2019.05.006. 22

Karol Piczak. Esc: Dataset for environmental sound classification. pages 1015–1018, 10 2015a. doi: 10.1145/2733373.2806390. 10

Karol Piczak. Environmental sound classification with convolutional neural networks. pages 1–6, 09 2015b. doi: 10.1109/MLSP.2015.7324337. 22

Payam Refaeilzadeh, Lei Tang, and Huan Liu. *Cross-Validation*, pages 532–538. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9. doi: 10.1007/978-0-387-39940-9_565. URL https://doi.org/10.1007/978-0-387-39940-9_565. 33

J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *22nd ACM International Conference on Multimedia (ACM-MM'14)*, pages 1041–1044, Orlando, FL, USA, Nov. 2014a. 37

Justin Salamon and Juan Pablo Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. *CoRR*, abs/1608.04363, 2016. URL http://arxiv.org/abs/1608.04363. 10, 22, 23, 32, 38

Justin Salamon, Christopher Jacoby, and Juan Bello. A dataset and taxonomy for urban sound research. *Proceedings - 22nd ACM International Conference on Multimedia*, 11 2014b. doi: 10.1145/2647868.2655045. 10

Justin Salamon, Duncan Macconnell, Mark Cartwright, Peter Li, and Juan Bello. Scaper: A library for soundscape synthesis and augmentation. 10 2017. doi: 10.1109/WASPAA.2017.8170052. 10

Nasir Saleem and Muhammad Khattak. Deep neural networks for speech enhancement in complex-noisy environments. *International Journal of Interactive Multimedia and Artificial Intelligence*, InPress:1, 01 2019. doi: 10.9781/ijimai.2019.06.001. 4

Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, and Mickey Aleksic. A quantization-friendly separable convolution for mobilenets. 03 2018. 30

Siddharth Sigtia, Adam M Stark, Sacha Krstulović, and Mark D Plumbley. Automatic environmental sound recognition: Performance versus computational cost. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(11):2096–2107, 2016. 8

Harsh Sinha and Pawan K. Ajmera. Interweaving convolutions : An application to audio classification harsh. In *2018 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). 2018.*, 2018. 23

Malcolm Slaney. Auditory toolbox: A matlab toolbox for auditory modeling work. technical report, version 2, interval research corporation, 1998. 1998. 12

Hongwei Song, Jiqing Han, and Shiwen Deng. A compact and discriminative feature based on auditory summary statistics for acoustic scene classification. *CoRR*, abs/1904.05243, 2019. URL http://arxiv.org/abs/1904.05243. 31

S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *Journal of the Acoustical Society of America*, 8:185–190, 1937. 12

Syntiant. Syntiant introduces second generation ndp120 deep learning processor for audio and sensor apps, 2020. URL https://www.syntiant.com/post/syntiant-introduces-second-generation\-ndp120-deep-learning-processor-for-audio-and-sensor-apps. 63

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015. 52

Artem Vasilyev. Cnn optimizations for embedded systems and fft. *Standford http://cs231n.stanford.edu/reports/2015/pdfs/tema8final.pdf*, 2015. 17

Haoren Wang, Haotian Shi, Xiaojun Chen, Liqun Zhao, Yixiang Huang, and Chengliang Liu. An improved convolutional neural network based approach for automated heartbeat classification. *Journal of Medical Systems*, 44, 02 2020. doi: 10.1007/s10916-019-1511-2. 4

P. Warden and D. Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers*. O'Reilly Media, Incorporated, 2019. ISBN 9781492052043. URL https://books.google.be/books?id=sB3mxQEACAAJ. 24

Shengyun Wei, Shun Zou, Feifan Liao, and Weimin Lang. A comparison on data augmentation methods based on deep learning for audio classification. *Journal of Physics: Conference Series*, 1453:012085, 01 2020. doi: 10.1088/1742-6596/1453/1/012085. 63

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. URL http://arxiv.org/abs/1707.01083. 22

Zhichao Zhang and Abbas Kouzani. Implementation of dnns on iot devices, volume = 32, journal = Neural Computing and Applications, doi = 10.1007/s00521-019-04550-w. 10 2019. 6